高等院校信息技术规划教材

HTML5网页开发实践教程

李洪波 主编
杨延村 崔光海 杨坤 赵峰 副主编

清华大学出版社

高等院校信息技术规划教材

HTML5 网页开发实践教程

李洪波　主编
杨延村　崔光海　杨　坤　赵　峰　副主编

清 华 大 学 出 版 社
北　京

内 容 简 介

本书面向 PC Web 应用和移动 Web 应用，包含理论教学和实验教学，各章明确知识目标、能力目标、素质目标，结合 HTML 基础、表单与 CSS、HTML 布局、JavaScript、HTML DOM、jQuery、JSON、HTML5、jQuery Mobile 和 Node.js 共十方面内容，分为理论基础与上机实验两个有机环节。上机实验分为上机实验样例讲解和实验任务布置两部分。上机实验样例包含实验目的、程序功能、源程序、实验步骤、程序分析、现象分析与运行结果等。实验任务由一个典型的画图 Web 应用程序分解而成，贯穿于各章，包括实验题目、实验目的、程序功能、实验类型、实验要求、实验环境、问题及解决办法、运行结果等部分。综合 Javascript、jQuery、JSON、HTML5、jQuery Mobile 知识，结合具体的实验样例和实验任务，同步阐述网页实用开发技术和面向对象应用程序开发技术。

本书适合作为高等院校计算机科学与技术、软件工程、网络工程、信息管理与信息系统专业本科生的教材，亦可作为社会培训机构的实训教材，还可作为网页开发工程技术人员的参考书。

图书在版编目(CIP)数据

HTML5 网页开发实践教程/李洪波主编. —北京：清华大学出版社，2017
(高等院校信息技术规划教材)
ISBN 978-7-302-48933-7

Ⅰ. ①H…　Ⅱ. ①李…　Ⅲ. ①超文本标记语言—程序设计—高等学校—教材　Ⅳ. ①TP312.8

中国版本图书馆 CIP 数据核字(2017)第 287982 号

责任编辑：张　玥
封面设计：常雪影
责任校对：白　蕾
责任印制：杨　艳

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 刷 者：北京富博印刷有限公司
装 订 者：北京市密云县京文制本装订厂
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张：**21　　**字　　数：**484 千字
版　　次：2017 年 12 月第 1 版　　**印　　次：**2017 年 12 月第1 次印刷
印　　数：1～1500
定　　价：45.00 元

产品编号：075798-01

前言

foreword

HTML5 新一代网页开发标准一经推出，以其跨平台、整合桌面 Web 应用和移动 Web 应用等优点受到广大厂商的青睐。当前主流 Web 浏览器，尤其是移动端 Web 浏览器都支持 HTML5 标准。HTML5 成为近年来最流行的 Web 应用前端开发工具，成为企业快速部署、到处运行 Web 应用的不二选择。运用 HTML5 进行 Web 应用前端开发，是最近几年需求广泛的职业岗位。HTML5 网页开发不但已作为很多高校信息类学科专业的必修课或选修课，也是社会 IT 类课程培训机构的热门课程之一。

本书按照由简单到复杂、由静态到动态、由单一到综合、由理论到实验、由低版本到高版本、由验证到设计、由 PC Web 到移动 Web 的技术演进、需求演进原则布局谋篇，以整合理论与实验，讲述网页制作与设计知识，突出跟踪、调试与部署方面的实践技能。

本书以 IT 企业对 Web 前端开发人员的知识要求、能力要求和素质要求为基础，以实践能力培养为宗旨，将知识目标、能力目标和素质目标自然明确地融入各章。各章按照由浅入深、由简到繁、由易到难的原则从前到后安排内容。各章内部按照理论基础知识、上机实验样例和实验任务脉络展开。本书设计一个画图 Web 应用程序，将其功能体系分解到各章中，作为实验任务的部署。

全书共分 10 章，依次为 HTML 基础、表单与 CSS、HTML 布局、JavaScript、HTML DOM、jQuery、JSON、HTML5、jQuery Mobile、Node.js。本书涉及浏览器端脚本程序调试、跟踪、异常处理技术、IIS Web 服务器发布技术、无线局域网配置技术、Windows 防火墙配置技术、Node.js 的 Windows 环境安装技术。本书所用的开发工具为 Microsoft Visual Studio 2010，但不限于 Microsoft Visual Studio 2010。.htm(或.html)格式文件占源程序文件 95%以上。对于.htm(或.html)格式文件，Microsoft Visual Studio 2010 仅起编辑器的作用。涉及.aspx 格式文件的地方需要 Microsoft Visual Studio 2010/2012/2013 开发环境支持。

选用本书作为教材,至少需要72学时,理论和实验各36学时。

本书具有以下特点:

(1) 理论与实践融为一体。各章均按照先理论基础作铺垫,然后讲解上机实验样例,最后部署实验任务的顺序进行。

(2) 对象与脚本有机统一。本书从第7章JSON开始直到第9章结束,进入用面向对象开发方法进行脚本程序开发的环节,实验任务对面向对象技术应用有明确要求。

(3) 能力与素质孕于知识。本书各章都有明确的知识目标、能力目标和素质目标,以本教材为支撑点,落实专业人才培养目标。

(4) 上机与实践指导完备。先进行上机实验样例讲解,而后进行实验任务的部署。上机实验流程明确清晰,包括实验目的、程序功能、源程序、代码分析、运行演示。实验任务部署具体,操作性强,包括目的、功能、性质、要求、原理、问题及解决方案、运行结果、程序分析。

(5) 教学配套资源完备。提供PPT课件、源程序、教学大纲、授课计划、试题、课程剖析。

(6) PC Web和移动Web应用集于一身。

本书由李洪波、杨延村、崔光海、杨坤、赵峰共同编写。全书主要由李洪波执笔、编程和统稿,杨延村和崔光海分别制作了第1章和第2章的插图,杨坤和赵峰完成了Web Socket部分内容。本书在编写过程中得到了清华大学出版社的大力支持,在此表示诚挚的感谢。

由于作者水平有限,书中难免有疏漏之处,恳请读者不吝赐教。

联系邮箱为738441242@qq.com。

编　者

2017年10月

目录 Contents

chapter 1

第1章 HTML基础

■ 知识目标

- 了解 Internet
- 了解 HTML 语言的特点
- 掌握 HTML 的文档结构
- 掌握 HTML 常用标签

■ 能力目标

- 能够利用浏览器浏览网页,查看文档结构,调试网页效果
- 能够利用 Visual Studio 2010 创建网站,新增 HTML 文件和编辑 HTML 文档
- 能够利用超链接进行任意跳转

■ 素质目标

- 将网页的基本结构归结于树
- 将查看器看到的元素抽象为对象

■ 教学重点

- HTML 文档结构
- 3 种形式的超链接
- 表格的设置

■ 建议学时

- 理论:3 学时
- 实验:3 学时

1.1 Internet 概述

Internet,译为因特网,又叫做国际互联网,是由使用公用语言互相通信的计算机连接而成的全球网络。一旦用户计算机连接到它的任何一个节点上,意味着该计算机已经连入 Internet 网上了。目前,Internet 的用户已经遍及全球,有数十亿人在使用 Internet,并且用户数还在以等比级数上升。

1. 网络应用

(1) 收发电子邮件

这是最早也是最广泛的网络应用。由于其具有费用低廉和快捷方便的特点，仿佛缩短了人与人之间的距离，不论身在异国他乡与朋友进行信息交流，还是联络工作，都如同对面聊天一样容易。

(2) 网络办公

网络的广泛应用会创造一种数字化的生活与工作方式，叫做小型家庭办公室。家庭将不再仅仅是人类社会生活的一个孤立单位，而是信息社会中充满活力的细胞。

(3) 上网浏览或冲浪

这是网络提供的最基本的服务项目。用户可以访问网上的任何网站，根据兴趣进行网上畅游，足不出户便可尽知天下事。

(4) 查询信息

网络是这个世界最大的资料库，利用一些供查询信息的搜索引擎，可以从浩如烟海的信息库中找到用户需要的信息。随着我国“政府上网”工程的发展，人们的一些日常事务几乎都可以在网络上完成。

(5) 电子商务

消费者借助网络，进入购物站点进行消费。网络上的购物站点建立在虚拟的数字化空间里，借助 Web 来展示商品，并利用多媒体特性加强商品的可视性、选择性。虽然目前的网络购物还不完善，但它已经实实在在地来到我们身边，使我们的生活多了一种选择。

(6) 丰富人们的闲暇生活方式

闲暇活动即非职业劳动的活动，它包括消遣娱乐型活动，如欣赏音乐、看电影、电视、跳舞、参加体育活动；发展型活动，包括学习文化知识、参加社会活动、从事艺术创造和科学发明活动等。但与网络有直接关系的闲暇生活一般包括闲暇教育、闲暇娱乐和闲暇交往。

(7) 网上交友聊天

随着网络聊天工具越来越普遍地应用于人们的生活之中，每个人都可以通过上网结交世界各地的网上朋友，相互交流思想，做到“海内存知己，天涯若比邻”。

(8) 其他应用

在现实世界中，人类活动的网络版俯拾即是，如网上点播、网上炒股、网上求职、艺术展览等。

总之，Internet 是大众创业万众创新的引擎，为各行各业的发展提供动力，把人类社会带入一个全新的数字历史阶段。Internet 已演变为信息经济的原动力和新引擎，演变为一个降低成本、提高生产力，并为各种新工作铺平道路的推土机。

未来高速网络连接及 Internet 连接将无处不在。家里、办公室里将有难以置信的网络连接速度，高速的网络连接将遍及各种环境。网络将完全制约人们的工作、生活及娱乐。

随着 Internet 在全球的普及和其在各个领域的广泛应用,工业时代那种以地缘为本的场地分割和垄断方式的国家和企业集团的模式会逐步被打破。我们面对的是一个统一的全球市场,经济将实现全球化。目前最突出的是网络环境下的经济模式——电子商务。

2. 网络功能

Internet 上有丰富的信息资源,可以通过 Internet 方便地寻求各种信息。可从两个来源寻求信息:人和计算机系统。在 Internet 上可以找到能够提供各种信息的人:教育家、科学家、工程技术专家、医生、营养学家、学生……以及有各种专长和爱好的人们。对于所有这些人,Internet 提供与处在同样情况下的其他人进行讨论和交流的渠道。实际上,几乎在所有可能想到的题目下都能找到讨论与交流的小组。或者,如果没有这样的讨论小组,我们还可以自己创建一个。

Internet 计算机存储的信息汇成了信息资源的大海洋。信息内容无所不包:有学科技术的各种专业信息,也有与大众日常工作与生活息息相关的信息;有严肃主题的信息,也有体育、娱乐、旅游、消遣和奇闻逸事一类的信息;有历史档案信息,也有现实世界的信息;有知识性和教育性的信息,也有消息和新闻的传媒信息;有学术、教育、产业和文化方面的信息,也有经济、金融和商业信息等。信息的载体涉及几乎所有媒体,如文档、表格、图形、影像、声音以及它们的合成信息。信息容量小到几行字符,大到一个图书馆。信息分布在世界各地的计算机上,以各种可能的形式存在,如文件、数据库、公告牌、目录文档和超文本文档等,而且这些信息还在不断地更新和变化中。可以说,这里是一个取之不尽、用之不竭的大宝库。

Internet 的另一种资源是计算机系统资源,包括连接在 Internet 各种网络上的计算机的处理能力、存储空间以及软件工具和软件环境。一般地说,要求使用计算机系统的 Internet 用户,如科学家、工程师、设计师、教师、学生或每一个普通用户,都可以远程登录某台目标计算机——只要这台计算机允许用户使用并建立了用户的登录账号。用户可以像使用自己的计算机一样使用它们。

进入 Internet 后,就可以利用各个网络和计算机上无穷无尽的资源,同世界各地自由通信和交换信息,以及去做通过计算机能做的各种各样的事情,享受 Internet 提供的各种服务。

(1) 高级浏览 WWW 服务

WWW(World Wide Web),简称 Web,是用户登录 Internet 后最常利用的功能。人们连入 Internet 后,有一半以上的时间都是在与各种各样的 Web 页面打交道。在基于 Web 的方式下,人们可以浏览、搜索、查询各种信息,发布自己的信息,与他人进行实时或者非实时的交流,可以游戏、娱乐、购物等。

(2) 电子邮件 E-mail 服务

在 Internet 上,电子邮件或称为 E-mail 系统是使用最多的网络通信工具,E-mail 已成为备受欢迎的通信方式。用户可以通过 E-mail 系统同世界上任何地方的朋友交换电子邮件。不论对方在哪里,只要也连入 Internet,他发送的信只需要几分钟就可以到达对

方的手中了。

(3) 远程登录 Telnet 服务

远程登录就是通过 Internet 进入和使用远距离的计算机系统,就像使用本地计算机一样。远端的计算机可以在同一间屋子里,也可以远在数千公里之外。它使用的工具是 Telnet。它在接到远程登录的请求后,就试图把用户所在的计算机同远端计算机连接起来。一旦连通,用户的计算机就成为远端计算机的终端。用户可以正式注册进入系统成为合法用户,执行操作命令,提交作业,使用系统资源。完成操作任务后,通过注销退出远端计算机系统,同时也退出 Telnet。

(4) 文件传输 FTP 服务

FTP(文件传输协议)是 Internet 上最早使用的文件传输程序。它同 Telnet 一样,使用户能登录到 Internet 的一台远程计算机,把其中的文件传送回自己的计算机系统,或者把本地计算机上的文件传送并装载到远方的计算机系统。利用这个协议,就可以下载免费软件,或者上传自己的主页了。

3. Internet 的特点

① 系统不与具体的专用网络相关联,用户可以在世界范围内的任何地点、任何时候方便地访问网络上面任何一个节点。

② 对用户的计算机和网络操作的要求很低。

③ 绝大部分报文是通过填写屏幕单证的方式形成的。

④ Internet 的带宽高。

⑤ Internet 的费用低。

1.2 HTML 文档基本结构

浏览 WWW 服务,前提是用户的计算机已接入 Internet,而且计算机上已安装浏览器。作为 Windows 操作系统的一部分,IE 浏览器随着操作系统一起安装到用户机器中。Firefox、Chrome、Opera 等个人计算机上的浏览器需要单独安装。安卓版的手机、浏览器也随着操作系统的安装而完成,无须单独安装。在浏览器输入域名或网址(一个全球唯一的 URL),把远方站点的 HTML 文档借助 HTTP 传输到本地进行浏览。HTML 网页是 Web 应用的前端,直接与用户接触。

1.2.1 基础知识

超文本标记语言或超文本链接标示语言(Hyper Text Markup Language,HTML),是目前网络上应用最为广泛的语言,也是构成网页文档的主要语言。HTML 独立于各种操作系统平台(UNIX、Windows 等)。自 1990 年以来,HTML 就一直被作为 World Wide Web 的信息表示语言,用于描述 Homepage 的格式设计和与 WWW 上其他 Homepage 的联结信息。使用 HTML 语言描述的文件,需要通过 WWW 浏览器显示出

效果。常用的 WWW 浏览器有 Chrome(谷歌)、Firefox(火狐)、IE8 及以上版本、Edge、Opera、Safari、QQ、360 等。

HTML 文本是由 HTML 命令组成的描述性文本,HTML 命令可以说明文字、图形、动画、声音、表格、链接等。HTML 的结构包括头部(head)、主体(body)两大部分,其中头部描述浏览器所需的信息,主体则包含所要说明的具体内容。HTML 网页文档的基本结构中主要包含 HTML、head、body 标记。

1. HTML 文件标记

<HTML>和</HTML>标记放在网页文档的最外层,表示这对标记间的内容是 HTML 文档。<HTML>放在文件开头,</HTML>放在文件结尾,在这两个标记中间嵌套其他标记。一个 HTML 文档仅有一对<HTML>和</HTML>。

2. head 文件头部标记

文件头用<head>和</head>标记,该标记出现在文件的起始部分。它是所有头部元素的容器。<head> 中的元素可以引用脚本、指示浏览器所用的样式表、提供元信息等。标记内的内容不在浏览器中显示,主要用来说明文件的有关信息,如文件标题、作者、编写时间、搜索引擎可用的关键词等。

可用在 head 部分标签有<base>、<link>、<meta>、<script>、<style>以及<title>。<title> 定义文档的标题,它是 head 部分中唯一必需的元素。在 head 标记内最常用的标记是网页主题标记,即 title 标记,它的格式为<title>网页标题</title>。

网页标题是提示网页内容和功能的文字,它将出现在浏览器的标题栏中。一个网页只能有一个标题,并且只能出现在文件的头部。

一个 HTML 文档仅有一对<head>和</head>。

3. body 文件主体标记

文件主体对用户可见,用<body>和</body>标记,它是 HTML 文档的主体部分。一个 HTML 文档仅有一对<body>和</body>。网页正文中的所有内容,包括文字、表格、图像、声音和动画等,都包含在这对标记对之间。通常的标记样式如下:

```
<body bgcolor="white" text="blue" link="yellow" alink="blue" vlink="pink" leftmargin="
120px" topmargin="150px">文档内容</BODY>
```

一个简单 HTML 文档如下:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xHTML1/DTD/xHTML1-transitional.dtd">
<HTML xmlns="http://www.w3.org/1999/xHTML">
    <head>
        <meta http-equiv="Content-Type" content="text/HTML; charset=gb2312" />
        <title>HTML5 文档结构</title>
    </head>
```

```
<body  bgcolor="white" text="blue" link="yellow" alink="blue" vlink= "pink"
topmargin="150px">
    静夜思
    李白
    床前明月光，
    疑是地上霜。
    举头望明月，
    低头思故乡。
</body>
</HTML>
```

综上所述，HTML是用来描述网页的一种语言。HTML不是一种编程语言，而是一种标记语言(markup language)。标记语言是一套标记标签(markup tag)。HTML使用标记标签来描述网页。HTML标记标签通常被称为HTML标签(HTML tag)。HTML标签是由尖括号包围的关键词，比如<HTML>。HTML标签通常成对出现，比如<body>和</body>。标签对中的第一个标签是开始标签，第二个标签是结束标签。开始和结束标签也被称为开放标签和闭合标签。HTML文档描述网页，包含HTML标签和纯文本，因此HTML文档也被称为网页。Web浏览器的作用是读取HTML文档，并以网页的形式显示它们。浏览器不会显示HTML标签，而是使用标签，以不同形式展示页面内容，使页面内容显示格式丰富多彩。

1.2.2 上机实验样例

1. 编辑HTML文档

可使用Windows桌面系统自带的记事本或写字板来编写简单的HTML，或使用专业的HTML编辑器来编辑复杂的HTML文档。专业的HTML编辑器有Adobe Dreamweaver、Front Page、Microsoft Expression Web、Coffee Cup HTML Editor和Microsoft Visual Studio集成开发环境等编辑器。本书选用Microsoft Visual Studio集成开发环境编辑HTML文档。步骤如下。

(1) 启动Microsoft Visual Studio 2010

Visual Studio 2010启动后的界面如图1.1所示。

(2) 新建网站

如图1.2所示，选择图1.1中的File菜单项，在弹出的快捷菜单中选择New，在右侧的列表中选择Web Site…，弹出图1.3所示的New Web Site对话框。在图1.3中选择ASP. NET Empty Web Site。采用默认的网站位置为C:\Users\lhb_dell\Documents\Visual Studio 2010\WebSites\WebSite2。如果需要更改为已经创建的文件夹，则需单击Browse按钮，通过资源浏览器加以选择。单击图1.3的OK按钮，则成功创建网站，如图1.4所示。网站的名称为网站位置的最底层目录名WebSite2。

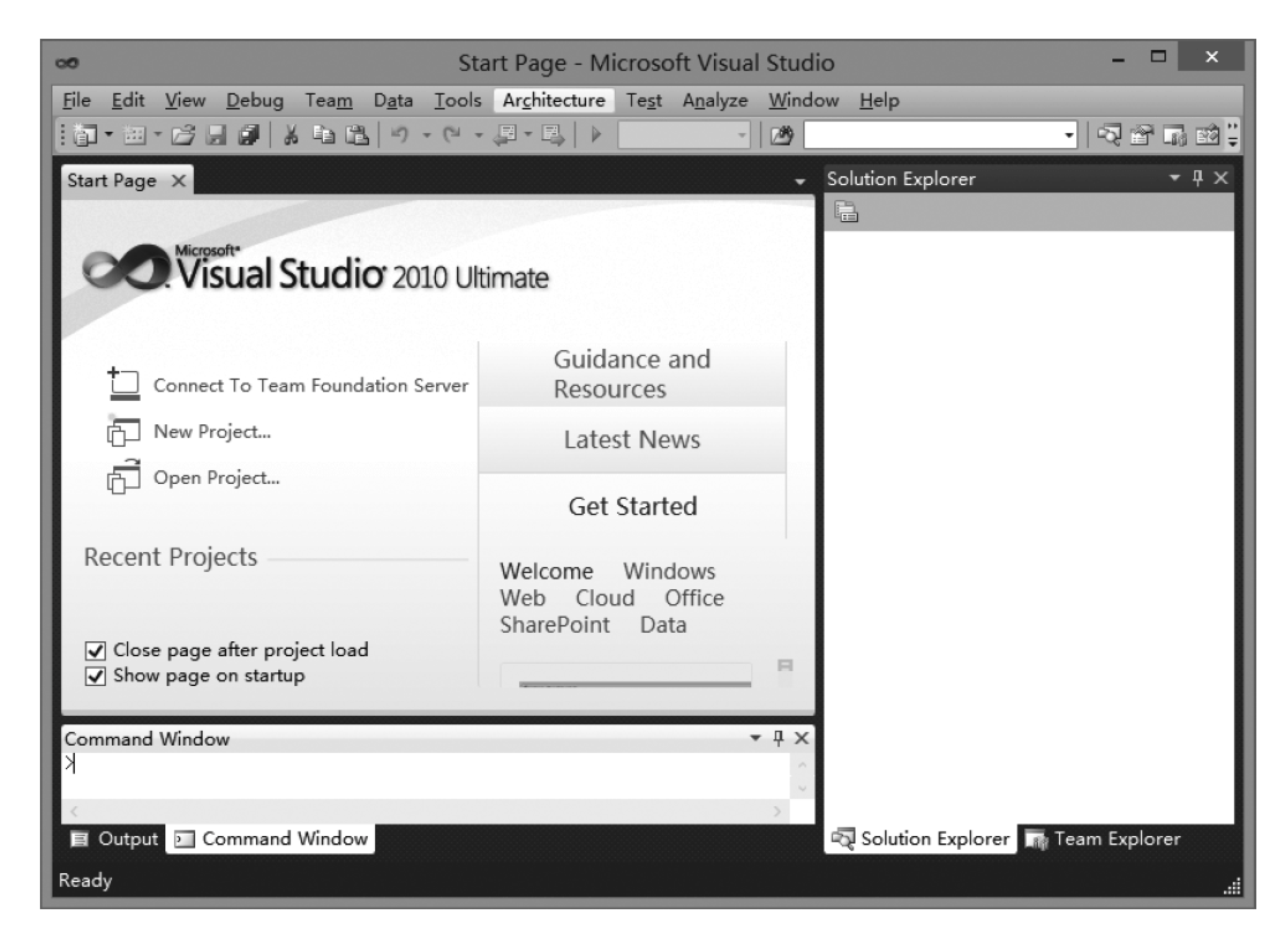


图 1.1 Visual Studio 2010 集成开发环境主界面

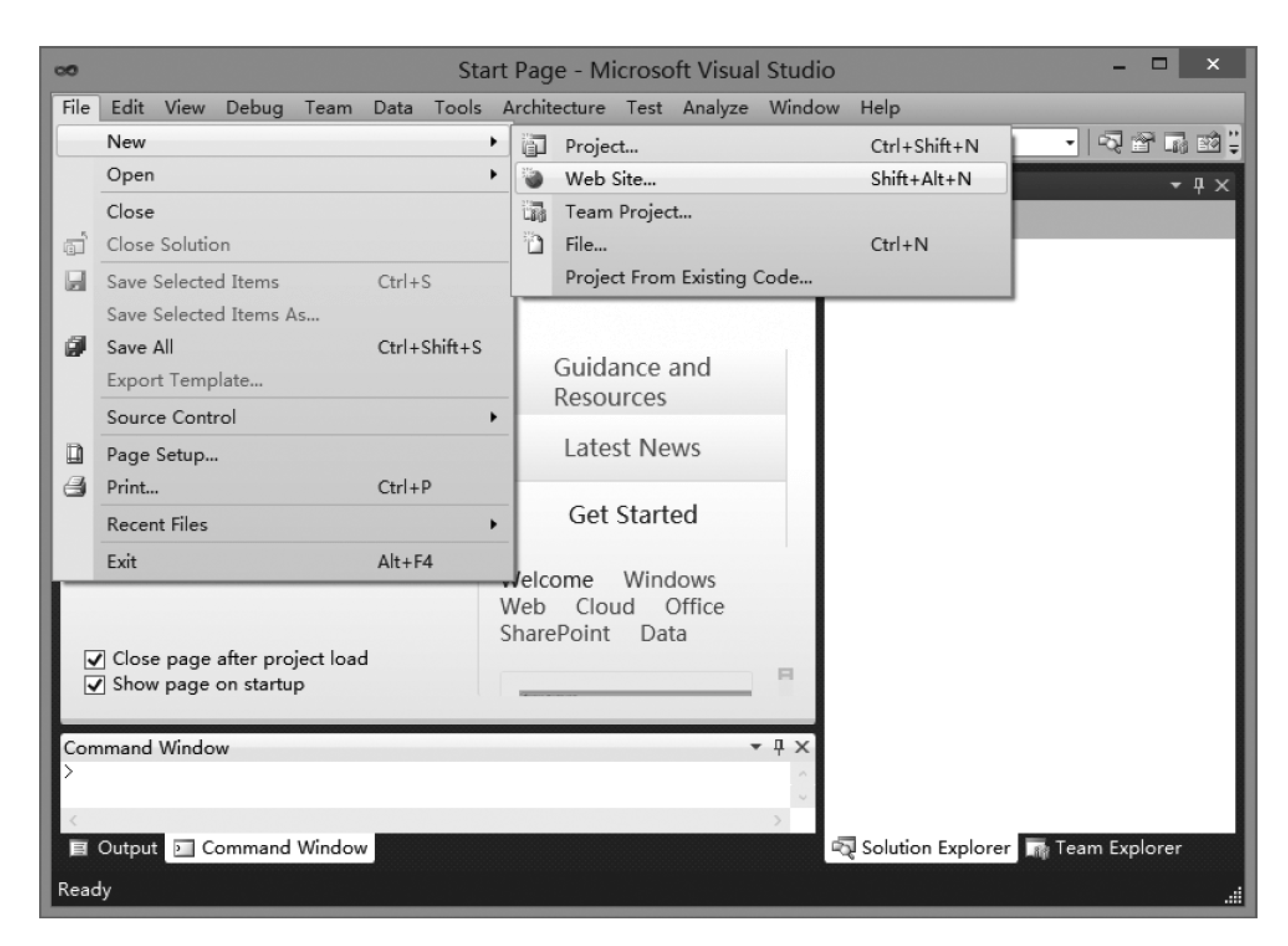


图 1.2 在 Visual Studio 2010 集成开发环境主界面中新建网站



图 1.3　New Web Site 对话框

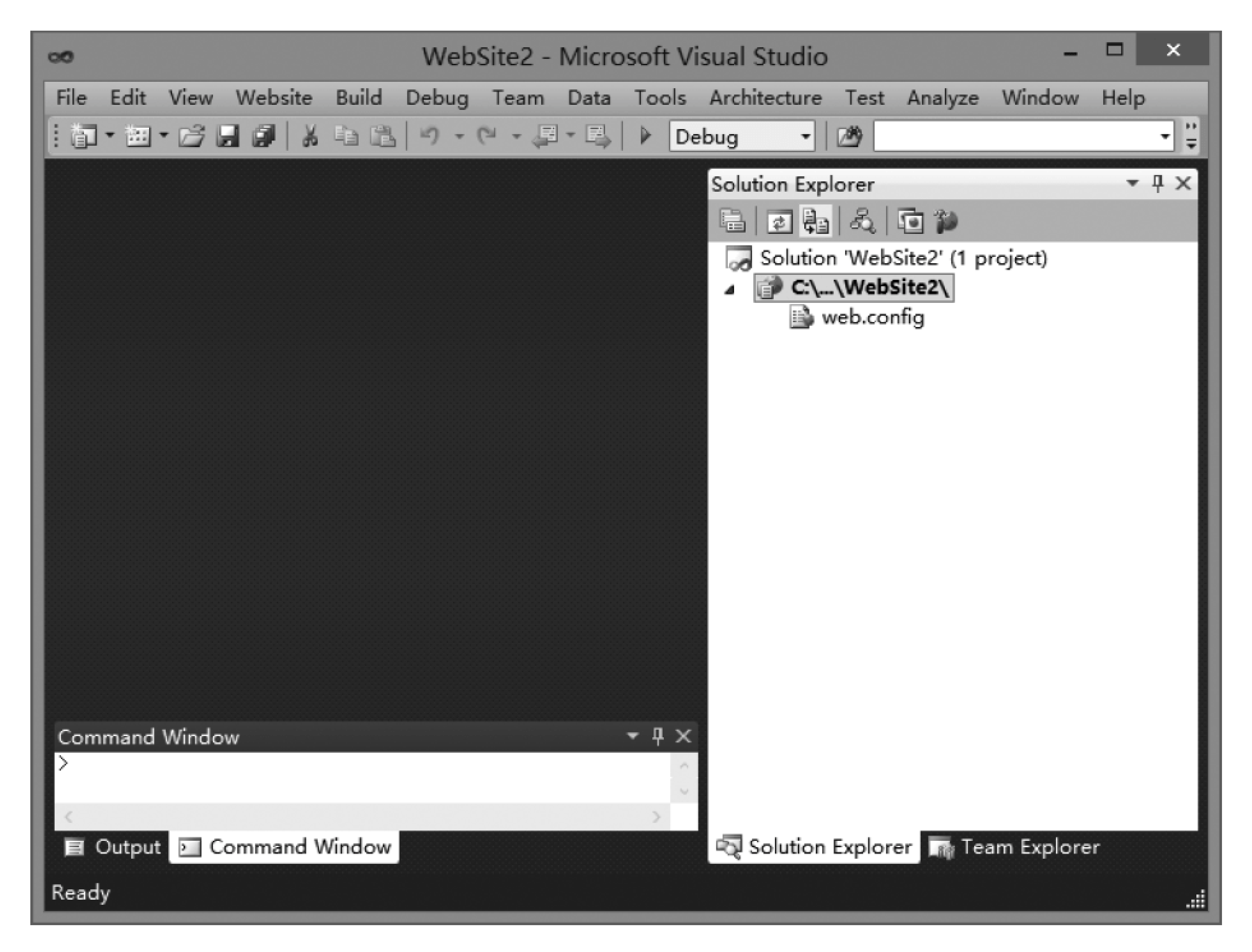


图 1.4　新建的 WebSite2 网站

(3) 添加 HTML 文档

在图 1.4 中选择 C:\...\WebSite2\站点，右击，弹出图 1.5 所示的快捷菜单。选择 Add New Item，弹出图 1.6 所示的对话框。在图 1.6 的 Sort by 中选择 HTML Page，在

Name 文本框中修改默认的文件名为 SimpleHTMLStructure.htm。单击 Add 按钮，打开图 1.7。

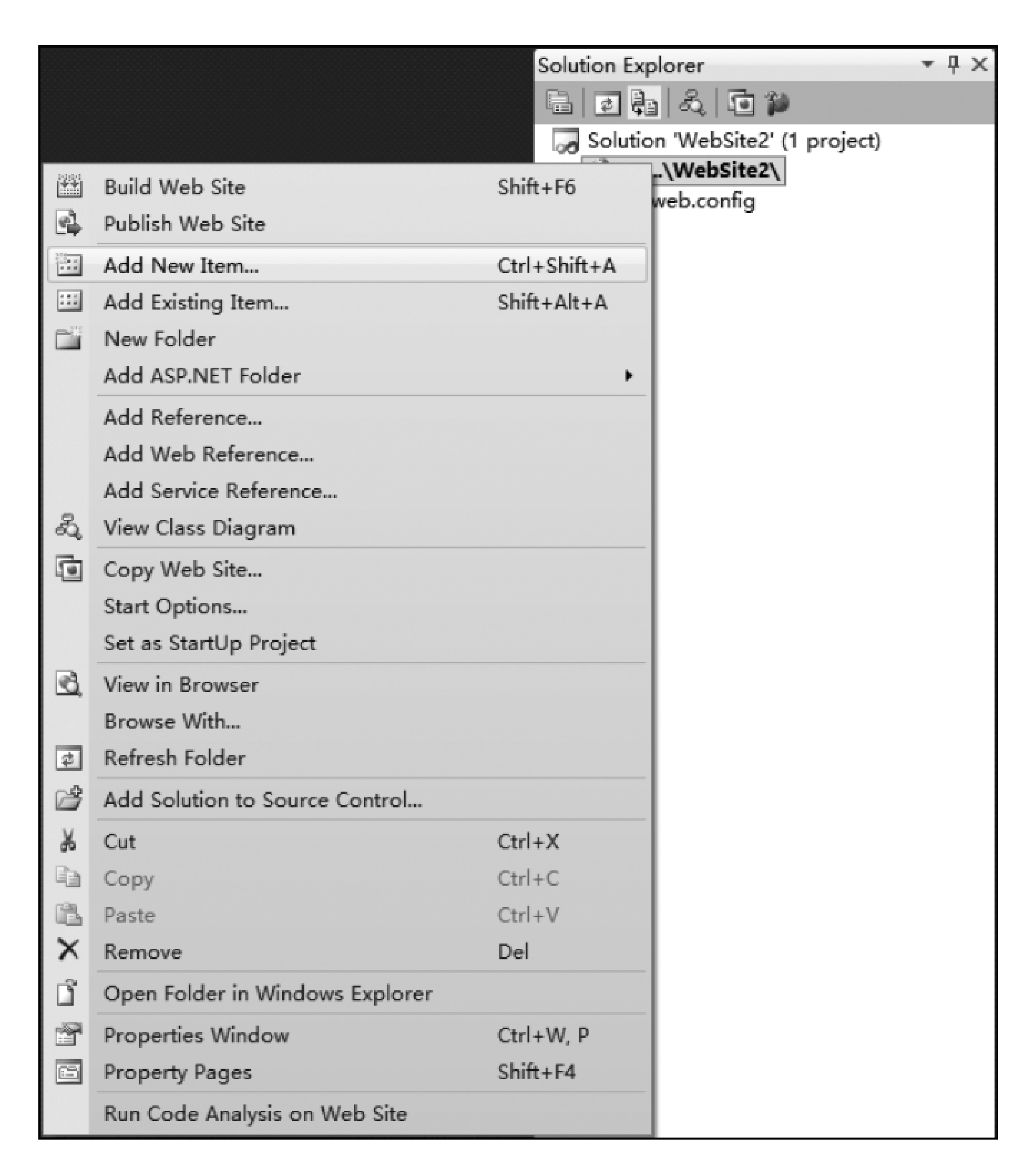


图 1.5 项目快捷菜单

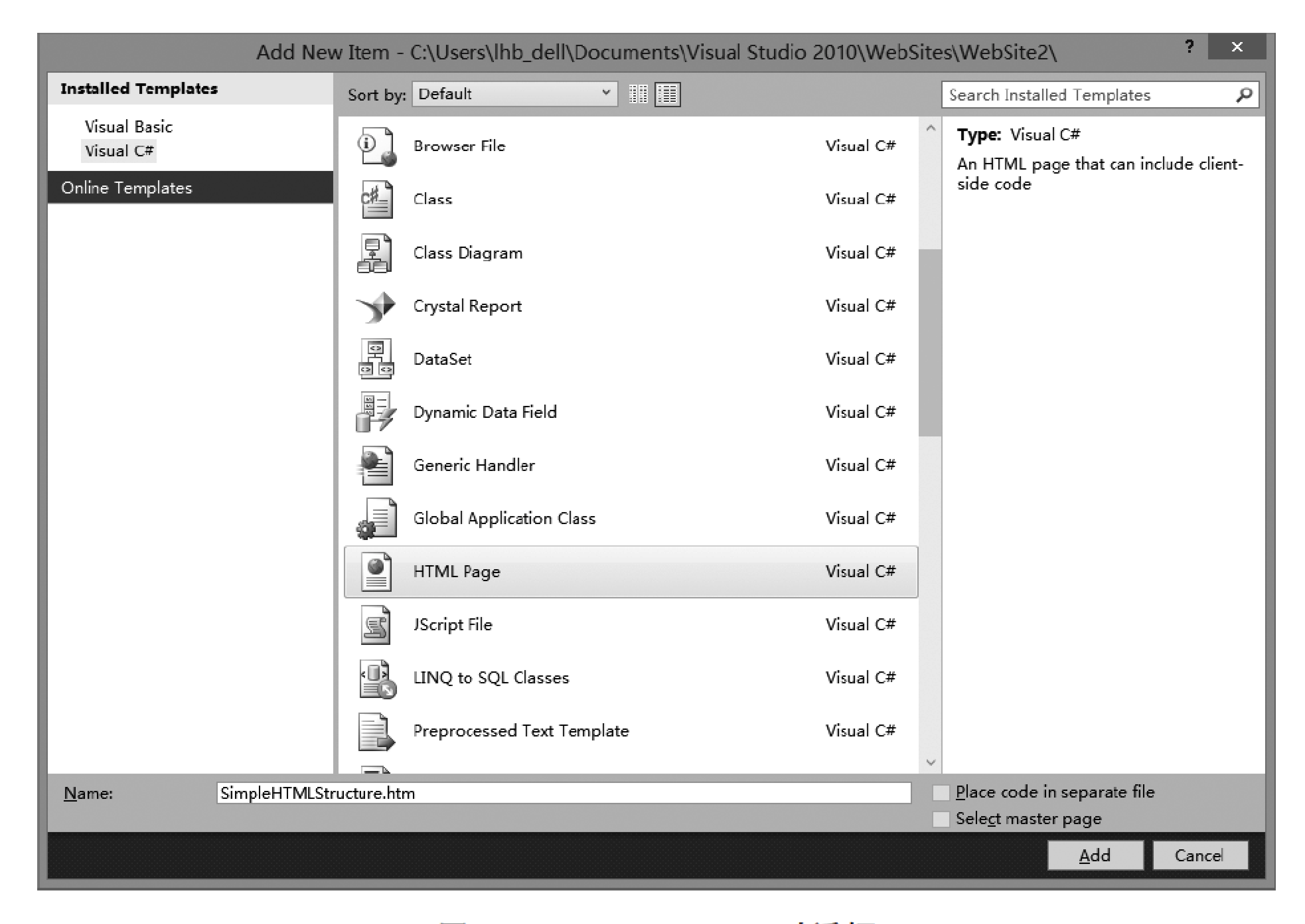


图 1.6 Add New Item 对话框

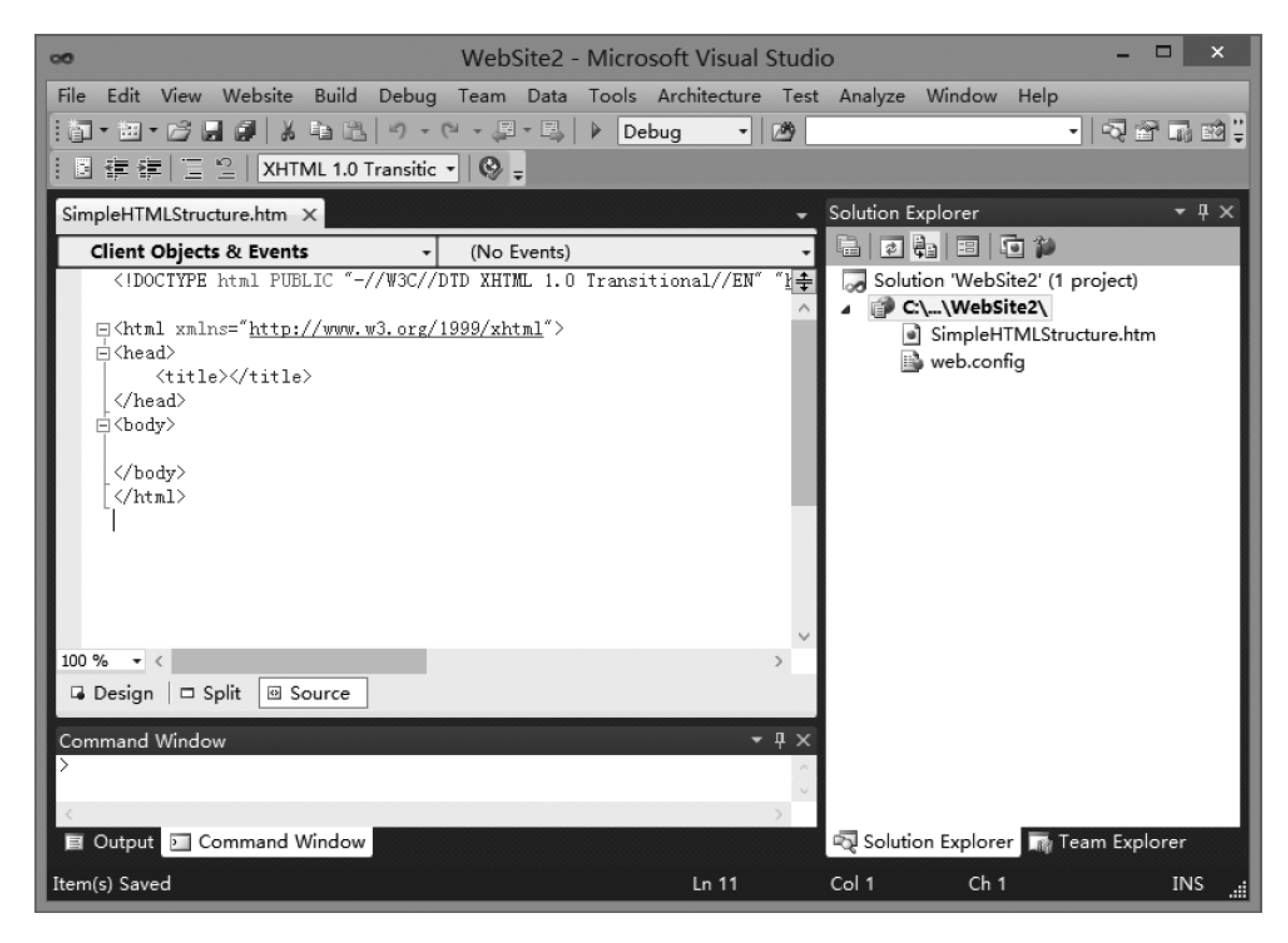


图 1.7　成功添加 SimpleHTMLStructure.htm 的视图

经过编辑的 HTML 文档如图 1.8 所示。

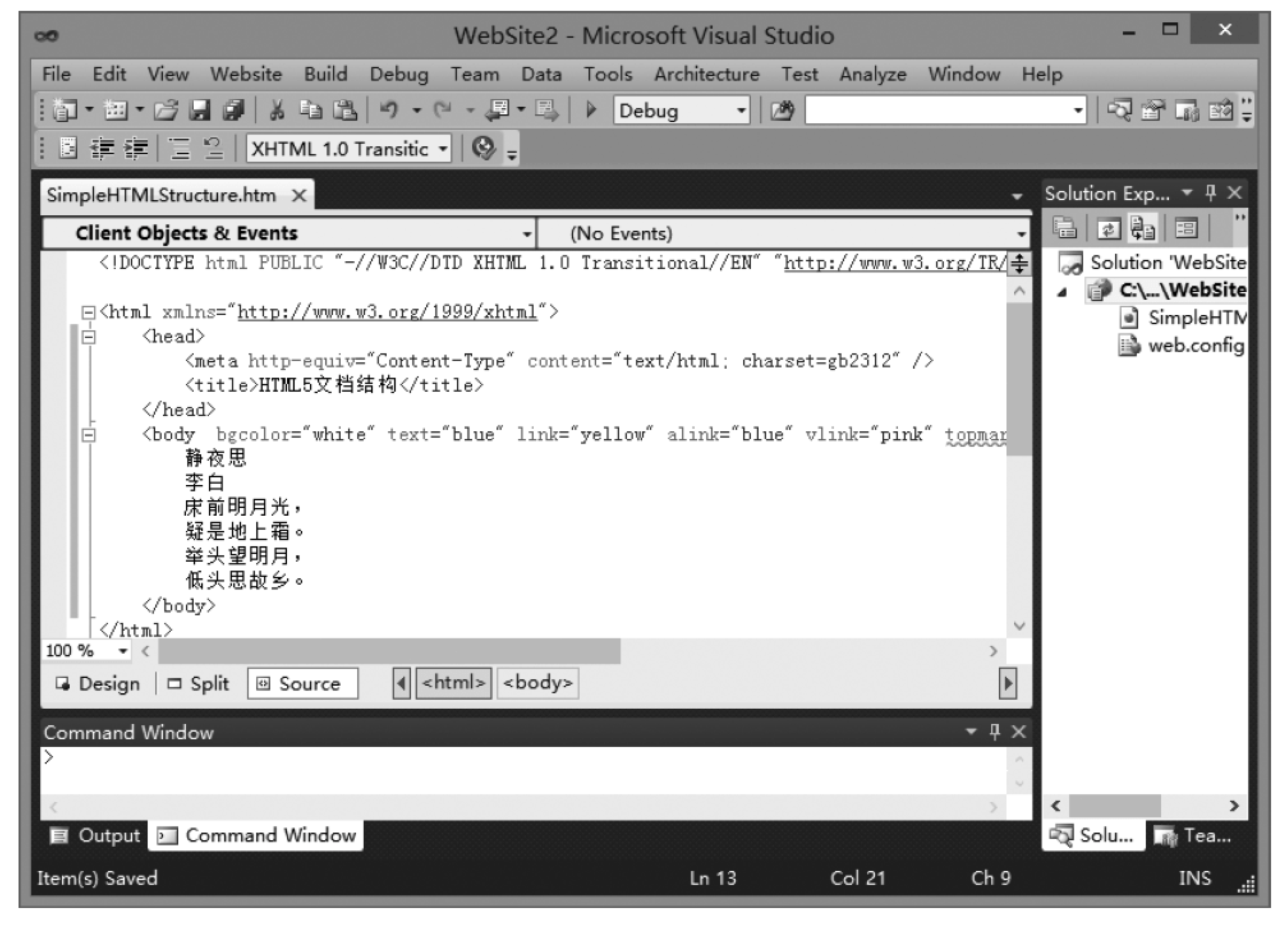


图 1.8　编辑后的 HTML 文档

2. 浏览 HTML 文档

浏览 HTML 文档有两种方式，一是在 Visual Studio 2010 集成开发环境中浏览，二是不在任何集成开发环境中浏览。

（1）在集成开发环境中浏览

如果没有设置默认浏览器，可以选中解决方案浏览器 WebSite2 项目内的 SimpleHTMLStructure.htm，右击弹出快捷菜单，如图 1.9 所示。选择 Browse With...，弹出设置默认浏览器对话框，选择 Firefox，单击 Set as Default 按钮，如图 1.10 所示。否则直接单击 View in Browser，弹出图 1.11 所示的网页。

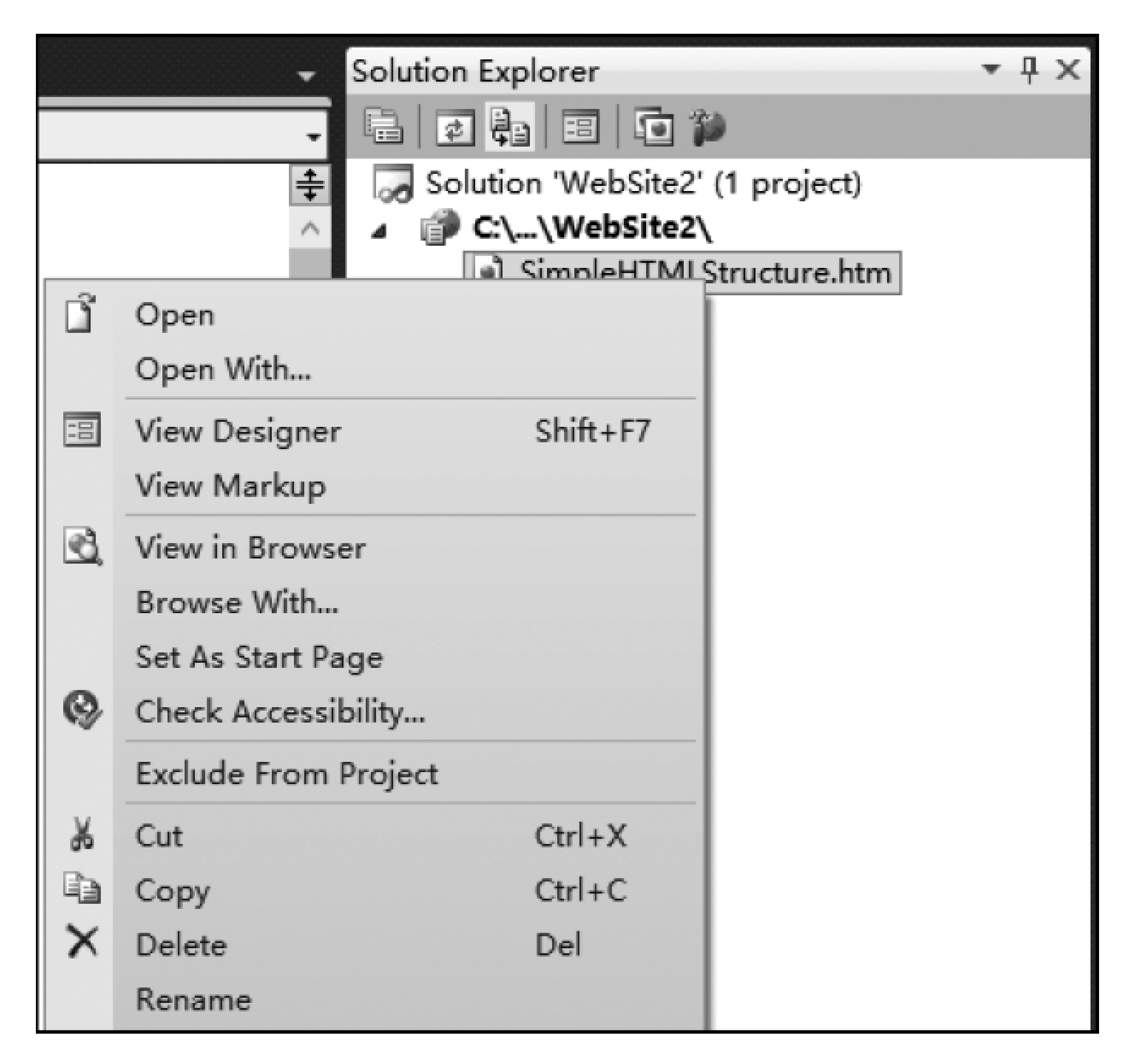


图 1.9　HTML 文档快捷菜单

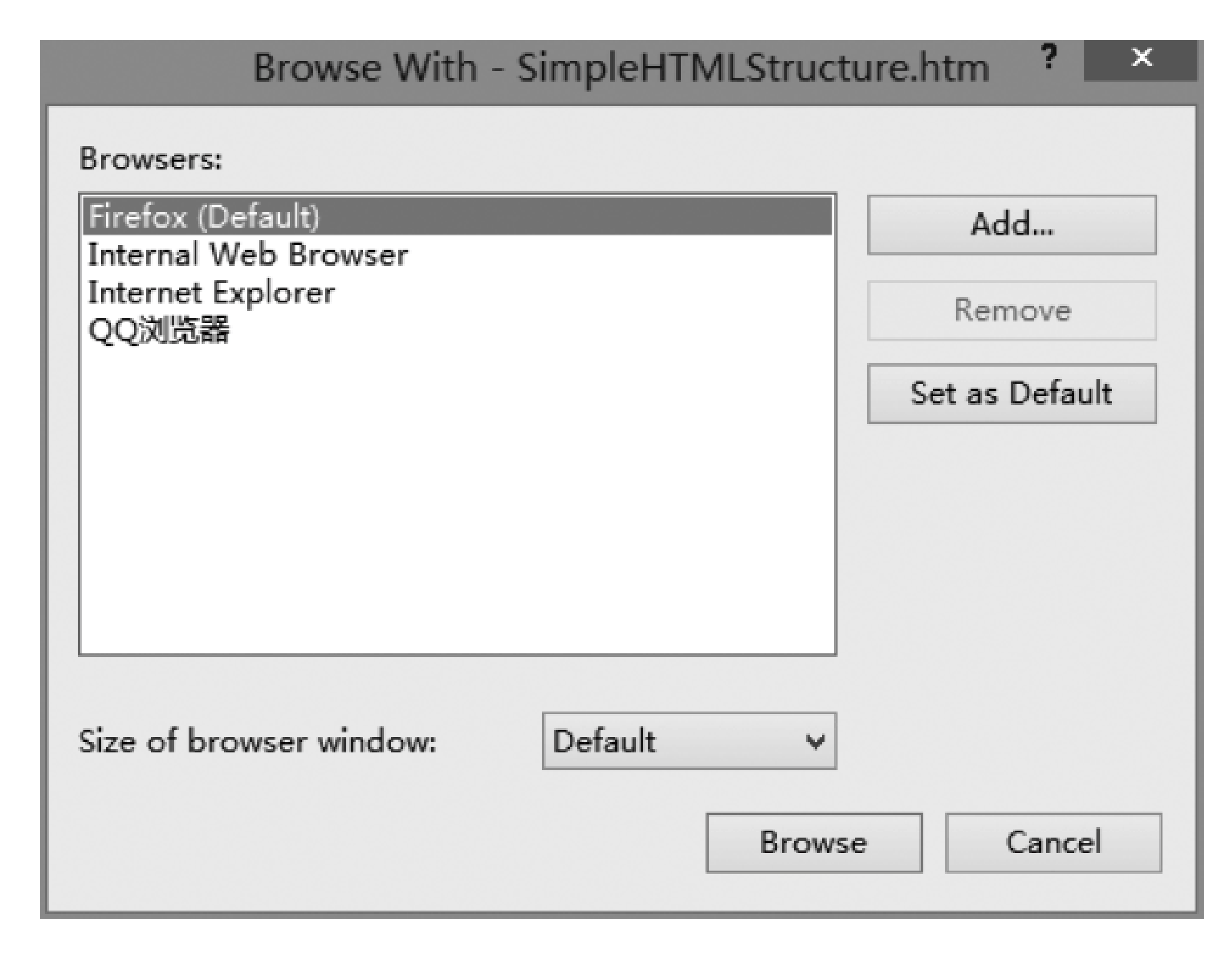


图 1.10　设置默认浏览器对话框



图 1.11　浏览 SimpleHTMLStructure.htm 网页

（2）在开发环境外浏览

在资源浏览器中选择 SimpleHTMLStructure.htm，右击，弹出快捷菜单，选择 Firefox，如图 1.12 所示，得到图 1.11 所示的网页。

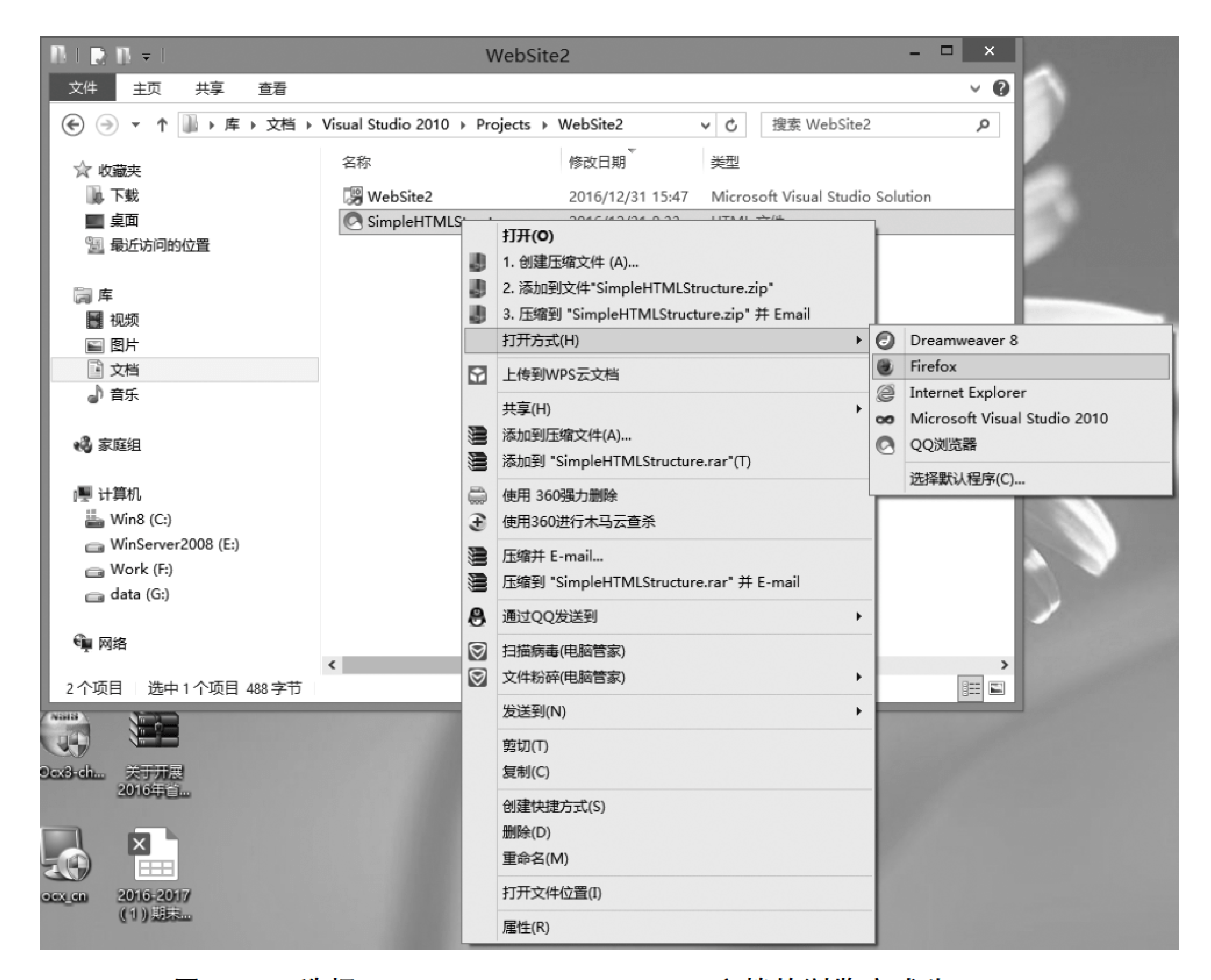


图 1.12　选择 SimpleHTMLStructure.htm 文档的浏览方式为 Firefox

3. 在 Firefox 浏览器中的查看器内查看 HTML 文档

按 F12 键激活 SimpleHTMLStructure.htm 网页，网页下部显示开发人员相关工具，选择"查看器"选项卡，默认选中 body 标签，如图 1.13 所示。双击标签内的区域，随即启动标签内相关属性或文本的编辑功能，双击《静夜思》文本区域，启动文本编辑功能，在"李白"前加入"作者-"，如图 1.14 所示。按回车键，则对编辑予以确认，并退出编辑，如图 1.15 所示。



图 1.13　激活"开发人员工具"后的网页



图 1.14　在浏览器中编辑 body 标签中的文本

值得注意的是，在浏览器中确认修改标签的属性或文本，仅对当前激活的浏览器一次有效，供调试观察效果所用，SimpleHTMLStructure.htm 文档源文件本身没有变化。

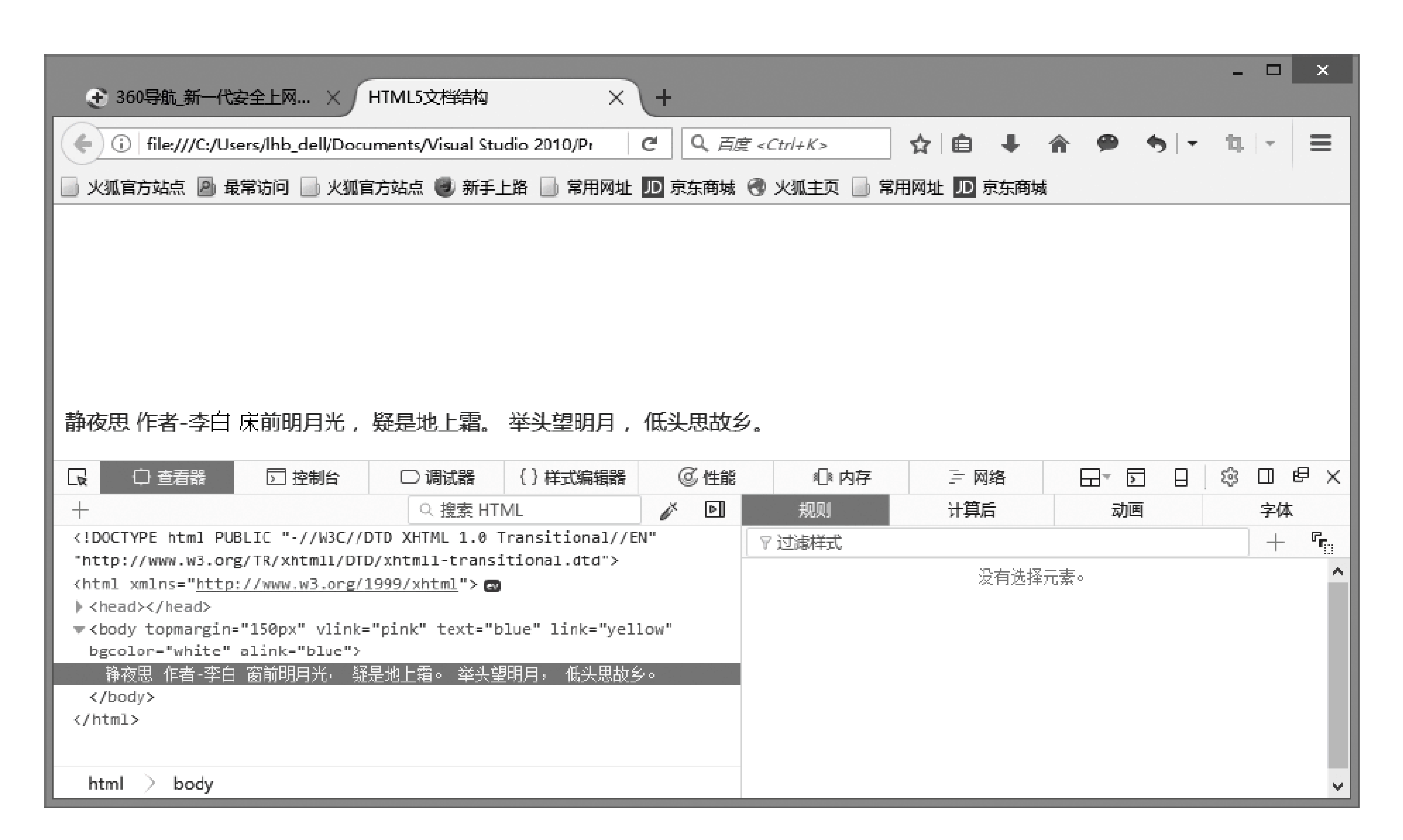


图 1.15 确认修改 body 标签中文本后的网页

再次按下 F12 键，则关闭开发人员工具。

1.2.3 实验观察与思考

（1）在 Visual Studio 2010 开发环境中，以个人学号和姓名建立及命名网站，网站的主题为：个人姓名＋“测试 HTML 文档结构”。新增 SimpleHTMLStructure.htm，在 Firefox 浏览器中的查看器修改图 1.13 所示的 body 背景色。

（2）仔细观察图 1.8 源文档中 body 标签内的文本部分，诗名、作者、4 句诗各占一行，共 6 行，但经浏览器显示后，却平拉为 1 行，分析原因并给出解决措施。

1.3 HTML 基本元素

一个网页由许多元素构成。HTML 定义了许多基本元素，供人们开发丰富的网页。

1.3.1 理论知识

HTML 标记标签通常被称为 HTML 标签（HTML tag）。HTML 标签是由尖括号包围的关键词，比如＜HTML＞。HTML 标签通常是成对出现的，比如＜body＞和＜/body＞，标签对中的第一个标签是开始标签，第二个标签是结束标签，开始和结束标签也被称为开放标签和闭合标签。HTML 文档是由 HTML 元素定义的。HTML 元素是从开始标签（start tag）到结束标签（end tag）的所有代码。元素的内容是开始标签与结束标签之间的内容。某些 HTML 元素具有空内容（empty content），空元素在开始标签中进行关闭（以开始标签的结束而结束）。大多数 HTML 元素可拥有属性。大多数 HTML 元素可以嵌套（可以包含其他 HTML 元素）。HTML 文档由嵌套的 HTML 元素构成。HTML 标签对大小写不敏感。

HTML 元素的基本格式如下：

```
<标签名 属性 1=值 1 属性 2=值 2 ……>元素内容</标签名>
```

HTML 标签可以拥有属性。属性提供了有关 HTML 元素的更多信息。属性总是以名称/值对的形式出现，比如 name="value"。属性在 HTML 元素的开始标签中规定。

属性值应该始终被包括在引号内。双引号是最常用的，不过使用单引号也没有问题。在某些个别的情况下，比如属性值本身就含有双引号，就必须使用单引号，例如：

```
name='Bill "HelloWorld" Gates'
```

1. 常用标签

(1) 标题

HTML 标题很重要。搜索引擎使用标题为网页的结构和内容编制索引。因为用户可以通过标题来快速浏览网页，所以用标题来呈现文档结构是很重要的。应该将 h1 用做主标题(最重要的)，其后是 h2(次重要的)，其次是 h3，以此类推。HTML 标题是通过<h1>～<h6>等标签进行定义的。<h1>定义最大的标题。<h6>定义最小的标题。浏览器会自动地在标题的前后添加空行。默认情况下，HTML 会自动地在块级元素前后添加一个额外的空行，比如段落、标题元素前后。

例 1-3-1-1 标题的简单示例。

```
<html>
    <body>
        <h1>This is heading 1</h1>
        <h2>This is heading 2</h2>
        <h3>This is heading 3</h3>
        <h4>This is heading 4</h4>
        <h5>This is heading 5</h5>
        <h6>This is heading 6</h6>
        <p>请仅仅把标题标签用于标题文本。不要仅仅为了产生粗体文本而使用它们。请使
        用其他标签或 CSS 代替。</p>
    </body>
</html>
```

(2) 段落

段落通过<p>标签定义。<p>是块级元素，浏览器会自动地在段落前后添加空行。使用空的段落标记<p></p>去插入一个空行，不如用
标签代替来插入空行。但不要用
标签去创建列表。不要忘记结束标签，即使忘了使用结束标签，大多数浏览器也会正确地将 HTML 显示出来。在大多数浏览器中，不加结束标签没问题，但不要依赖这种做法。忘记使用结束标签，有时会产生意想不到的结果和错误。

在未来的 HTML 版本中，不允许省略结束标签。通过结束标签来关闭 HTML 是一种经得起未来考验的 HTML 编写方法。清楚地标记某个元素在何处开始，并在何处结

束，不论对网页设计者还是对浏览器来说，都会使代码更容易理解。

例 1-3-1-2 段落应用的简单示例。

```
<html>
    <body>
        <h1>春晓</h1>
        <p>
            春眠不觉晓，
              处处闻啼鸟。
                夜来风雨声，
                  花落知多少。
        </p>
        <p>注意，浏览器忽略了源代码中的排版（省略了多余的空格和换行）。</p>
    </body>
</html>
```

（3）折行

在不产生一个新段落的情况下进行换行，使用
标签。
元素是一个空的 HTML 元素。由于关闭标签没有任何意义，因此它可以没有结束标签。在 XHTML、XML 以及未来的 HTML 版本中，不允许使用没有结束标签（闭合标签）的 HTML 元素。即使
在所有浏览器中的显示都没有问题，使用
也是更长远的保障。

例 1-3-1-3 折行的简单示例。

```
<html>
    <body>
        <p>
            To break<br />lines<br />in a<br />paragraph,<br />use the br tag.
        </p>
    </body>
</html>
```

（4）水平线

<hr/>标签在 HTML 页面中创建水平线。hr 元素可用于分隔内容。

例 1-3-1-4 一个水平线的简单示例。

```
<html>
    <body>
        <p>hr 标签定义水平线:</p>
        <hr />
        <p>这是段落。</p>
        <hr />
        <p>这是段落。</p>
        <hr />
        <p>这是段落。</p>
    </body>
```

```
</html>
```

(5) 注释

通过如下语法向 HTML 源代码添加注释：

```
<!--在此处写注释-->
```

开始标签中有一个惊叹号，但是结束标签中没有。浏览器不显示注释，但能够帮助记录设计者的 HTML 文档。可以利用注释在 HTML 中放置通知和提醒信息。注释对于 HTML 纠错也大有帮助，因为可以一次注释一行或多行 HTML 代码，以搜索错误。

例 1-3-1-5 使用注释的示例。

```
<!DOCTYPE html>
<html>
    <body>
        <!--这是一段注释。注释不会在浏览器中显示。-->
        <p>这是一段普通的段落。</p>
    </body>
</html>
```

(6) 链接

HTML 使用超级链接与网络上的另一个文档相连。几乎可以在所有网页中找到链接。单击链接可以从一张页面跳转到另一张页面。超链接可以是一个字、一个词或者一组词，也可以是一幅图像，可以单击这些内容跳转到新的文档或当前文档中的某个部分。当把鼠标指针移动到网页中的某个链接上时，箭头会变为一只小手。通过使用<a>标签在 HTML 中创建链接。

有两种使用 <a> 标签的方式：

- href 属性：创建指向另一个文档的链接。
- name 属性：创建文档内的书签。

HTML 链接最简单的语法如下：

```
<a href="url">Link text</a>
```

① href 属性。

href 属性规定链接目标。开始标签和结束标签之间的文字作为超级链接来显示。链接目标必须为 url 地址，如果没有给出具体路径，则默认路径和当前页的路径相同。链接到的文件也分为几种情况：如果为 HTML 文件，则在当前浏览器中直接打开；如果为可执行文件(.exe 文件)，则直接执行或下载，网页提供的下载文件就是用这种特性实现的；如果为文本文件，如 word 格式的文件，则在浏览器中打开此文件，并可以进行编辑加工。

例 1-3-1-6 href 属性实现超链接的示例。

```
<html>
    <body>
```

```
        <p>
        <a href="/index.html">本文本</a>是一个指向本网站中的一个页面的链接。</p>
        <p><a href="http://www.microsoft.com/">本文本</a>是一个指向万维网上的页面
        的链接。</p>
    </body>
</html>
```

② target 属性。

使用 target 属性,可以定义被链接的文档在何处显示。target 属性的枚举值如下所列:

• _blank

```
<a href="example.html" target="_blank">example</a>
```

浏览器会另开一个新窗口显示 example.html 文档。

• _parent

```
<a href="example.html" target="_parent">example</a>
```

指向父 frameset 文档。

• _self

```
<a href="example.html" target="_self">example</a>
```

把文档调入当前页框。

• _top

```
<a href="example.html" target="_top">example</a>
```

去掉所有页框,并用 document.html 取代 frameset 文档。

③ name 属性。

name 属性规定锚(anchor)的名称。可以使用 name 属性创建 HTML 页面中的书签。书签不会以任何特殊方式显示,它对读者是不可见的。当使用命名锚(named anchors)时,可以创建直接跳至该命名锚(比如页面中某个小节)的链接,这样使用者就无须不停地滚动页面来寻找所需要的信息了。

命名锚的语法:

```
<a name="label">锚(显示在页面上的文本)</a>
```

锚的名称可以是任何名字。可以使用 id 属性来替代 name 属性,命名锚同样有效。

例 1-3-1-7 锚应用的简单示例。

首先,在 HTML 文档中命名锚(创建一个书签):

```
<a name="tips">基本的注意事项-有用的提示</a>
```

然后,在同一个文档中创建指向该锚的链接:

```
<a href="#tips">有用的提示</a>
```

也可以在其他页面中创建指向该锚的链接：

```
<a href="http://www.w3school.com.cn/html/html_links.asp#tips">有用的提示</a>
```

在上面的代码中，将＃符号和锚名称添加到 URL 的末端，则可以直接链接到 tips 这个命名锚。

(7) 图像

可以在 HTML 文档中显示图像。在 HTML 中，图像由<img>标签定义。<img>是空标签，它只包含属性，并且没有闭合标签。要在页面上显示图像，需要使用源属性 src。源属性值是图像的 URL 地址。

定义图像的语法是：

```
<img src="url" />
```

URL 指存储图像的位置。如果名为 boat.gif 的图像位于 www.w3school.com.cn 的 images 目录中，那么其 URL 为 http://www.w3school.com.cn/images/boat.gif。

浏览器将图像显示在文档中图像标签出现的地方。如果将图像标签置于两个段落之间，那么浏览器会首先显示第一个段落，然后显示图片，最后显示第二段。

alt 属性用来为图像定义一串预备的可替换的文本。替换文本属性的值是用户定义的。例如<img src="boat.gif" alt="Big Boat">。在浏览器无法载入图像时，替换文本属性告诉读者失去的信息。此时，浏览器将显示这个替代性的文本而不是图像。为页面上的图像都加上替换文本属性是个好习惯，这样有助于更好地显示信息，对于那些使用纯文本浏览器的人来说，这是非常有用的。

假如某个 HTML 文件包含 10 个图像，为了正确地显示这个页面，需要加载 11 个文件。加载图片是需要时间的，所以慎用图片。

例 1-3-1-8 显示一张图片的简单示例。

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head><title></title></head>
    <body>
        <p>一幅图像:<img src="Images/0.jpg" width="128" height="128" alt="灯具"/></
        p>
        <p>一幅动画图像:
            <img src="http://www.w3school.com.cn/i/eg_cute.gif" width="50" height="50"
            alt=""/>
        </p>
        <p>请注意,插入动画图像的语法与插入普通图像的语法没有区别。</p>
    </body>
</html>
```

浏览器显示如图 1.16 所示。



图 1.16　例 1-3-1-8 的浏览器显示

① 背景图片。

例 1-3-1-9　背景图片示例。

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head><title></title></head>
    <body background="Images/0.jpg">
        <h3>图像背景</h3>
        <p>gif 和 jpg 文件均可用做 HTML 背景。</p>
        <p>如果图像小于页面,图像会进行重复。</p>
    </body>
</html>
```

浏览器显示如图 1.17 所示。



图 1.17　例 1-3-1-9 的浏览器显示

② 对齐方式。

例 1-3-1-10 对齐方式使用示例。

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
    </head>
    <body>
        <h2>未设置对齐方式的图像:</h2>
        <p>图像<img src="http://www.w3school.com.cn/i/eg_cute.gif" alt="默认"/>在文本中默认对齐</p>
        <h2>已设置对齐方式的图像:</h2>
        <p>图像<img src="http://www.w3school.com.cn/i/eg_cute.gif" align="bottom" alt="底部"/>在文本中底部对齐</p>
        <p>图像<img src="http://www.w3school.com.cn/i/eg_cute.gif" align="middle" alt="中部"/>在文本中中部对齐</p>
        <p>图像<img src="http://www.w3school.com.cn/i/eg_cute.gif" align="top" alt="顶部"/>在文本中顶部对齐</p>
        <p>请注意,bottom 对齐方式是默认的对齐方式。</p>
    </body>
</html>
```

浏览器显示如图 1.18 所示。



图 1.18 例 1-3-1-10 的浏览器显示

③ 浮动。

例 1-3-1-11 浮动使用示例。

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
    </head>
    <body>
        <p>
        <img src="http://www.w3school.com.cn//i/eg_cute.gif" align="left">
        带有图像的一个段落。图像的 align 属性设置为 "left"。图像将浮动到文本的左侧。
        </p>
        <p>
        <img src="http://www.w3school.com.cn//i/eg_cute.gif" align="right">
        带有图像的一个段落。图像的 align 属性设置为 "right"。图像将浮动到文本的右侧。
        </p>
    </body>
</html>
```

浏览器显示如图 1.19 所示。



图 1.19 例 1-3-1-11 的浏览器显示

④ 调整大小。

例 1-3-1-12 调整大小的示例。

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head><title></title></head>
    <body>
        <img src="http://www.w3school.com.cn/i/eg_mouse.jpg" width="50" height="50"
        alt=""/><br/>
        <img src="http://www.w3school.com.cn/i/eg_mouse.jpg" width="100" height="100"
        alt=""/><br/>
        <p>通过改变 img 标签的 "height" 和 "width" 属性的值,您可以放大或缩小图像。</
        p>
    </body>
</html>
```

浏览器显示如图 1.20 所示。



图 1.20　例 1-3-1-12 的浏览器显示

⑤ 制作图像链接。

例 1-3-1-13　图像链接的示例。

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head><title></title></head>
    <body>
        <p>
            您也可以把图像作为链接来使用：
            <a href="hello.htm">
                <img border="0" src="http://www.w3school.com.cn/i/eg_
                buttonnext.gif" alt=""/>
            </a>
        </p>
    </body>
</html>
```

浏览器显示如图 1.21 所示。



图 1.21　例 1-3-1-13 的浏览器显示

(8) 列表。

HTML 支持有序、无序和定义列表。

① 无序列表。

无序列表是一个项目的列表，此列项目使用粗体圆点（典型的小黑圆圈）进行标记。无序列表始于 <ul> 标签。每个列表项始于 。

例 1-3-1-14 无序列表的示例。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
</head>
<body>
    <h4>一个无序列表:</h4>
    <ul>
      <li>咖啡</li>
      <li>茶</li>
      <li>牛奶</li>
    </ul>
</body>
</html>
```

浏览器显示如图 1.22 所示。



图 1.22 无序列表的浏览器显示

列表项内部可以使用段落、换行符、图片、链接以及其他列表等。

② 有序列表。

同样，有序列表也是一列项目，列表项目使用数字进行标记。有序列表始于<ol>标签。每个列表项始于<li>标签。start 属性指明序号起始值。列表项内部可以使用段落、换行符、图片、链接以及其他列表等。

例 1-3-1-15 有序列表的示例。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
```

```
xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
</head>
<body>
    <ol>
      <li>咖啡</li>
      <li>牛奶</li>
      <li>茶</li>
    </ol>
    <ol start="50">
      <li>咖啡</li>
      <li>牛奶</li>
      <li>茶</li>
    </ol>
</body>
</html>
```

浏览器显示如图 1.23 所示。



图 1.23　有序列表的浏览器显示

(9) 表格

表格由<table>标签定义。每个表格均有若干行(由<tr>标签定义),每行被分割为若干单元格(由<td>标签定义)。字母 td 指表格数据(table data),即数据单元格的内容。数据单元格可以包含文本、图片、列表、段落、表单、水平线、表格等。如果不定义边框属性,表格将不显示边框。有时这很有用,但是大多数时候,需要显示边框。使用边框属性(border)可以显示一个带有边框的表格。

例 1-3-1-16　表格示例。

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
```

```
</head>
<body>
    <table border="1">
        <tr>
            <td>row 1, cell 1</td>
            <td>row 1, cell 2</td>
        </tr>
        <tr>
            <td>row 2, cell 1</td>
            <td>row 2, cell 2</td>
        </tr>
    </table>
</body>
</html>
```

浏览器显示如图 1.24 所示。

row 1, cell 1	row 1, cell 2
row 2, cell 1	row 2, cell 2

图 1.24　例 1-3-1-16 的浏览器显示

① 表头。

表格的表头使用<th>标签定义。大多数浏览器会把表头显示为粗体居中的文本。

例 1-3-1-17　带表头的表格。

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
</head>
<body>
    <table border="1">
        <tr>
            <th>Heading</th>
            <th>Another Heading</th>
        </tr>
        <tr>
            <td>row 1, cell 1</td>
            <td>row 1, cell 2</td>
        </tr>
        <tr>
            <td>row 2, cell 1</td>
            <td>row 2, cell 2</td>
        </tr>
    </table>
</body>
</html>
```

浏览器显示如图 1.25 所示。

Heading	Another Heading
row 1, cell 1	row 1, cell 2
row 2, cell 1	row 2, cell 2

图 1.25　例 1-3-1-17 的浏览器显示

② 空单元格。

在一些浏览器中，没有内容的表格单元显示效果

不好。如果某个单元格是空的(没有内容),浏览器可能无法显示出这个单元格的边框。为了避免这种情况,在空单元格中添加一个空格占位符( ),就可以将边框显示出来。

③ 标题。

caption 元素定义表格标题。caption 标签必须紧随 table 标签之后。只能对每个表格定义一个标题。通常标题居中于表格之上。

例 1-3-1-18 带有标题的表格示例。

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
    </head>
    <body>
        <h4>这个表格有一个标题以及粗边框:</h4>
        <table border="6">
            <caption align="top">我的标题</caption>
            <tr>
              <td>100</td>
              <td>200</td>
              <td>300</td>
            </tr>
            <tr>
              <td>400</td>
              <td>500</td>
              <td>600</td>
            </tr>
        </table>
    </body>
</html>
```

这个表格有一个标题,以及粗边框:

我的标题

100	200	300
400	500	600

图 1.26 例 1-3-1-18 的浏览器显示

浏览器显示如图 1.26 所示。

④ 跨列。

跨列由属性 colspan 设置。

例 1-3-1-19 跨列表格示例。

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
    </head>
    <body>
        <table border="2" align="center">
            <caption>创建表格</caption>
            <tr>
                <th colspan="3">第一学期</th>
```

```
                    <th colspan="3">第二学期</th>
                </tr>
                <tr>
                    <td>数学</td>
                    <td>科学</td>
                    <td>英语</td>
                    <td>数学</td>
                    <td>科学</td>
                    <td>英语</td>
                </tr>
                <tr>
                    <td>98</td>
                    <td>95</td>
                    <td>80</td>
                    <td>95</td>
                    <td>87</td>
                    <td>88</td>
                </tr>
            </table>
        </body>
</html>
```

浏览器显示如图 1.27 所示。

创建表格

第一学期			第二学期		
数学	科学	英语	数学	科学	英语
98	95	80	95	87	88

图 1.27　例 1-3-1-19 的浏览器显示

⑤ 跨行。

跨行由 rowspan 属性设置。

例 1-3-1-20　跨行表格示例。

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
    </head>
    <body>
        <table border="1" align="center">
            <caption>创建表格</caption>
            <tr>
                <th colspan="2"></th>
                <th>螺母</th>
                <th>螺栓</th>
                <th>锤子</th>
            </tr>
            <tr>
                <td rowspan="3">第一季度</td>
                <td>一月</td><td>2500</td><td>1000</td><td>1240</td>
            </tr>
```

```
            <tr>
                <td>二月</td><td>3000</td><td>2500</td><td>4000</td>
            </tr>
            <tr>
                <td>三月</td><td>3200</td><td>1000</td><td>2400</td>
            </tr>
        </table>
    </body>
</html>
```

浏览器显示如图 1.28 所示。

⑥ 表格内标签。

表格的单元格可以含有标签，这非常重要，因此表格不但内容显示样式丰富，而且可用于布局。



创建表格

		螺母	螺栓	锤子
第一季度	一月	2500	1000	1240
	二月	3000	2500	4000
	三月	3200	1000	2400

图 1.28　例 1-3-1-20 的浏览器显示

例 1-3-1-21　表格单元格含有标签的示例。

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
    </head>
    <body>
        <table border="1">
        <tr>
          <td>
              <p>这是一个段落。</p>
              <p>这是另一个段落。</p>
          </td>
          <td>这个单元包含一个表格：
              <table border="1">
                  <tr><td>A</td><td>B</td></tr>
                  <tr><td>C</td><td>D</td></tr>
              </table>
          </td>
        </tr>
        <tr>
            <td>这个单元包含一个列表：
                <ul>
                    <li>苹果</li>
                    <li>香蕉</li>
                    <li>菠萝</li>
                </ul>
            </td>
            <td>HELLO</td>
        </tr>
        </table>
    </body>
```

```
</html>
```

浏览器显示如图 1.29 所示。

⑦ 单元格边距(cellpadding)。

使用 cellpadding 来创建单元格内容与其边框之间的空白。

例 1-3-1-22 设置 padding 属性的表格示例。

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
    </head>
    <body>
        <h4>没有 cellpadding:</h4>
        <table border="1">
            <tr><td>First</td><td>Row</td></tr>
            <tr><td>Second</td><td>Row</td></tr>
        </table>
        <h4>带有 cellpadding:</h4>
        <table border="1" cellpadding="10">
            <tr><td>First</td><td>Row</td></tr>
            <tr><td>Second</td><td>Row</td></tr>
        </table>
    </body>
</html>
```

浏览器显示如图 1.30 所示。

这是一个段落。 这是另一个段落。	这个单元包含一个表格： A B C D
这个单元包含一个列表： • 苹果 • 香蕉 • 菠萝	HELLO

图 1.29 例 1-3-1-21 的浏览器显示

没有 cellpadding：

First	Row
Second	Row

带有 cellpadding：

First	Row
Second	Row

图 1.30 例 1-3-1-22 的浏览器显示

⑧ 单元格间距。

使用 cellspacing 增加单元格之间的距离。

例 1-3-1-23 设置 spacing 属性的表格示例。

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
    </head>
```

```
    <body>
        <h4>没有 cellspacing:</h4>
        <table border="1">
            <tr><td>First</td><td>Row</td></tr>
            <tr><td>Second</td><td>Row</td></tr>
        </table>
        <h4>带有 cellspacing:</h4>
        <table border="1" cellspacing="10">
            <tr>
              <td>First</td><td>Row</td>
            </tr>
            <tr>
              <td>Second</td><td>Row</td>
            </tr>
        </table>
    </body>
</html>
```

浏览器显示如图 1.31 所示。

⑨ 表格背景。

表格背景颜色由属性 bgcolor 设置,表格背景图片由其属性 background 设置。

例 1-3-1-24 设置 bgcolor 属性的表格示例。

```
<h4>背景颜色:</h4>
<table border="1" bgcolor="red">
    <tr><td>First</td><td>Row</td></tr>
    <tr><td>Second</td><td>Row</td></tr>
</table>
<h4>背景图像:</h4>
<table border="1" background="http://www.w3school.com.cn/i/eg_bg_07.gif">
    <tr><td>First</td><td>Row</td></tr>
    <tr><td>Second</td><td>Row</td></tr>
</table>
```

浏览器显示如图 1.32 所示。

没有 cellspacing:

First	Row
Second	Row

带有 cellspacing:

First	Row
Second	Row

图 1.31 例 1-3-1-23 的浏览器显示

背景颜色:

First	Row
Second	Row

背景图像:

First	Row
Second	Row

图 1.32 例 1-3-1-24 的浏览器显示

⑩ 单元格背景。

单元格也有背景颜色属性 bgcolor 和背景图片属性 background，分别用于设置背景颜色和背景图片。

例 1-3-1-25 设置单元各背景示例。

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
    </head>
    <body>
        <h4>单元格背景:</h4>
        <table border="1">
            <tr>
              <td bgcolor="red">First</td><td>Row</td>
            </tr>
            <tr>
              <td background="http://www.w3school.com.cn/i/eg_bg_07.gif">
              Second</td>
              <td>Row</td>
            </tr>
        </table>
    </body>
</html>
```

单元格背景：

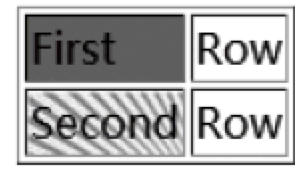


图 1.33 例 1-3-1-25 的浏览器显示

浏览器显示如图 1.33 所示。

(10) 文本格式化

文本格式化标签如表 1.1 所示。

表 1.1 文本格式化标签

标　签	描　述	标　签	描　述
<b>	定义粗体文本	<strong>	定义加重语气
<big>	定义大号字	<sub>	定义下标字
<em>	定义着重文字	<sup>	定义上标字
<i>	定义斜体字	<ins>	定义插入字
<small>	定义小号字	<del>	定义删除字

(11) 预格式化

使用 pre 标签对空行和空格进行控制。

例 1-3-1-26 具有预格式化标签的文档示例。

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
```

```
    </head>
    <body>
        <pre>
        这是
        预格式文本。
        它保留了      空格
        和换行。
        </pre>
        <p>pre 标签很适合显示计算机代码：</p>
        <pre>
        for i=1 to 10
            print i
        next i
        </pre>
    </body>
</html>
```

浏览器显示如图 1.34 所示。

```
这是
预格式文本。
它保留了      空格
和换行。

pre 标签很适合显示计算机代码：

for i = 1 to 10
    print i
next i
```

图 1.34　例 1-3-1-26 的浏览器显示

(12) 特殊字符

特殊字符如表 1.2 所示。

表 1.2　特殊字符表

字符	命名	字符	命名	字符	命名
Α	Α	Β	Β	Γ	Γ
Δ	Δ	Ε	Ε	Ζ	Ζ
Η	Η	Θ	Θ	Ι	Ι
Κ	Κ	Λ	Λ	Μ	Μ
Ν	Ν	Ξ	Ξ	Ο	Ο
Π	Π	Ρ	Ρ	Σ	Σ
Τ	Τ	ϒ	Υ	Φ	Φ
Χ	Χ	Ψ	Ψ	Ω	Ω
α	α	β	β	γ	γ
δ	δ	ε	ε	ζ	ζ
η	η	θ	θ	ι	ι
κ	κ	λ	λ	μ	μ
ν	ν	ξ	ξ	ο	ο
π	π	ρ	ρ	ζ	ς
σ	σ	τ	τ	υ	υ
φ	φ	χ	χ	ψ	ψ
ω	ω	□	ϑ	□	ϒ

续表

字符	命名	字符	命名	字符	命名
□	ϖ	•	•	…	…
′	′	″	″	‾	‾
/	⁄	?	℘	?	ℑ
?	ℜ	™	™	?	ℵ
←	←	↑	↑	→	→
↓	↓	↔	↔	?	↵
?	⇐	?	⇑	?	⇒
?	⇓	?	⇔	?	∀
∂	∂	?	∃	?	∅
?	∇	∈	∈	?	∉
?	∋	Π	∏	Σ	∑
-	−	?	∗	√	√
∝	∝	∞	∞	∠	∠
∧	∧	∨	∨	∩	∩
∪	∪	∫	∫	∴	∴
?	∼	?	≅	≈	≈
≠	≠	≡	≡	⩽	≤
⩾	≥	?	⊂	?	⊃
?	⊄	?	⊆	?	⊇
⊕	⊕	?	⊗	⊥	⊥
?	⋅	?	⌈	?	⌉
?	⌊	?	⌋	◇	◊
?	♠	♣	♣	♥	♥
?	♦			i	¡
¢	¢	£	£	¤	¤
¥	¥	≫	¦	§	§
¨	¨	©	©	ª	ª
《	«	¬	¬		
®	®	¯	¯	°	°
±	±	^{2}	²	^{3}	³
'	´	μ	µ	"	"
<	<	>	>	'	

2. Style 属性

Style 属性用于改变 HTML 元素的样式。它提供了一种改变所有 HTML 元素样式的通用方法。样式是 HTML4 引入的，它是一种新的首选的改变 HTML 元素样式的方式。修改 HTML 样式，可通过使用 Style 属性直接将样式添加到 HTML 元素，或者间接地在独立的样式表中(CSS 文件)进行定义。

在 HTML4 中，有若干的标签和属性是被废弃的。被废弃(Deprecated)的意思是在未来版本的 HTML 和 XHTML 中将不支持这些标签和属性。避免使用的被废弃的标签和属性如表 1.3 所示。

表 1.3　避免使用的被废弃的标签和属性

标　　签	描　　述	属　　性	描　　述
<center>	定义居中的内容	align	定义文本的对齐方式
<font>和<basefont>	定义 HTML 字体	bgcolor	定义背景颜色
<s>和<strike>	定义删除线文本	color	定义文本颜色
<u>	定义下画线文本		

① 背景颜色。

background-color 属性为元素定义了背景颜色。

例 1-3-1-27　设置背景样式的示例。

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
    </head>
    <body style="background-color:yellow">
        <h2 style="background-color:red">This is a heading</h2>
        <p style="background-color:green">This is a paragraph.</p>
    </body>
</html>
```

浏览器显示如图 1.35 所示。

② 字体、颜色和属性。

font-family、color 以及 font-size 属性分别定义元素中文本的字体系列、颜色和字体尺寸。

例 1-3-1-28　设置字体、前景色和字号的示例。

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
    </head>
    <body>
```

```
        <h1 style="font-family:verdana">A heading</h1>
        <p style="font-family:arial;color:red;font-size:20px;">A paragraph.</p>
    </body>
</html>
```

浏览器显示如图 1.36 所示。



图 1.35 例 1-3-1-27 的浏览器显示



图 1.36 例 1-3-1-28 的浏览器显示

③ 文本对齐。

text-align 属性规定了元素中文本的水平对齐方式。

例 1-3-1-29 文本对齐示例。

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
    </head>
    <body>
        <h1 style="text-align:center">This is a heading</h1>
        <p>The heading above is aligned to the center of this page.</p>
    </body>
</html>
```

浏览器显示如图 1.37 所示。



图 1.37 例 1-3-1-29 的浏览器显示

1.3.2 上机实验样例

1. 任意位置的超链接

超链接分为站内和站外两种情况。站内又分为同一网页和不同网页两种情形。同一网页内的跳转需要给出锚点标签的ID或NAME。不同网页可能处于相同目录或不同目录。对于不同目录,需要回退到不同目录的公共目录,而后转入另一个目录的目标网页甚至锚点元素ID或NAME。回退用../来实现。站外需要给出协议、域名、路径、网页以及目标标签ID或NAME信息。下面的实验演示任意位置的超链接。实验步骤如下。

(1) 添加"1-3-2-1-超链接"目录

在资源管理器中添加"1-3-2-1-超链接"目录,如图1.38所示。



图1.38 超链接目录

(2) 启动Visual Studio 2010

(3) 打开网站"1-3-2-1-超链接"

打开网站"1-3-2-1-超链接",如图1.39所示。

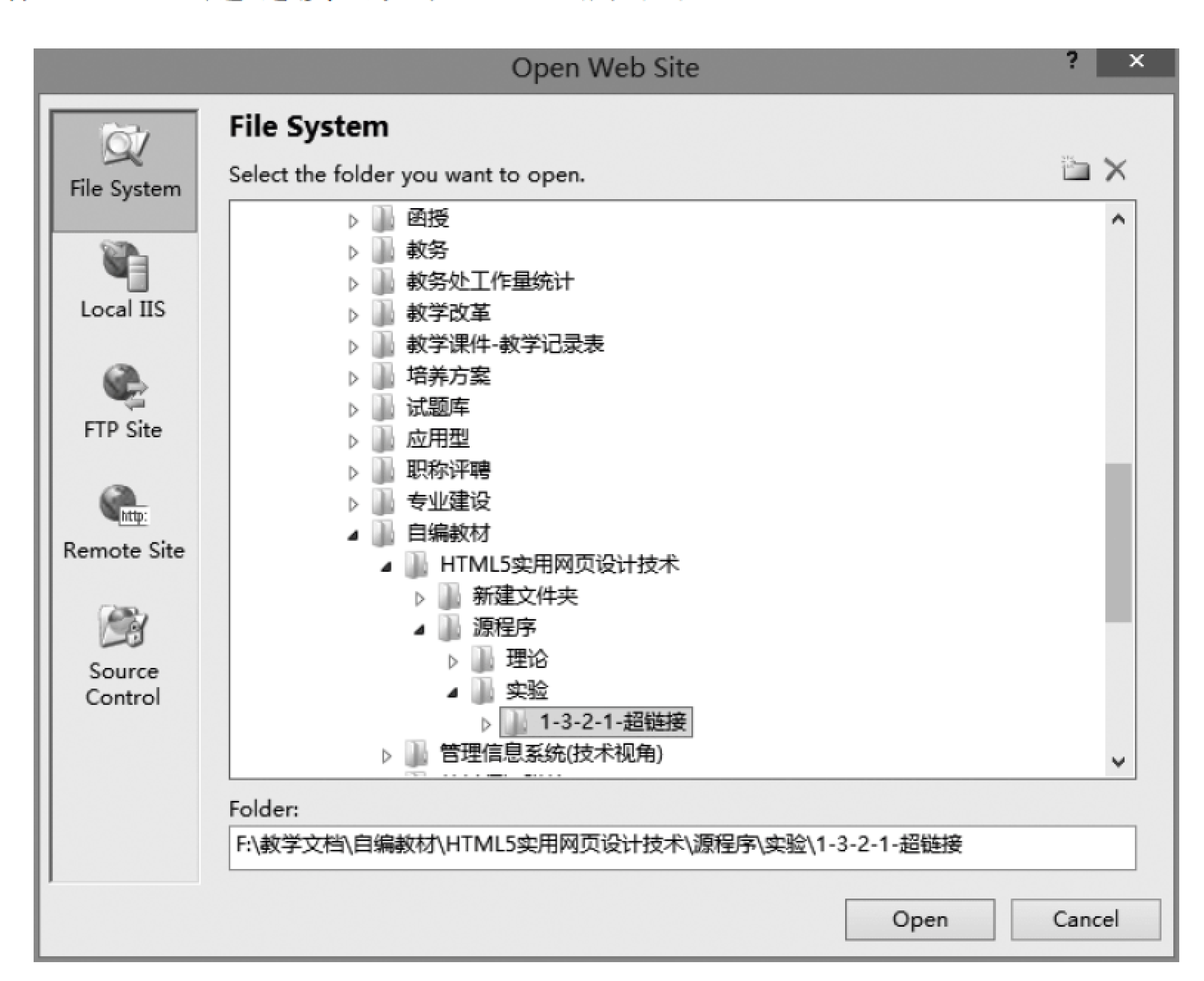


图1.39 打开网站"1-3-2-1-超链接"

打开超链接的窗口如图 1.40 所示。

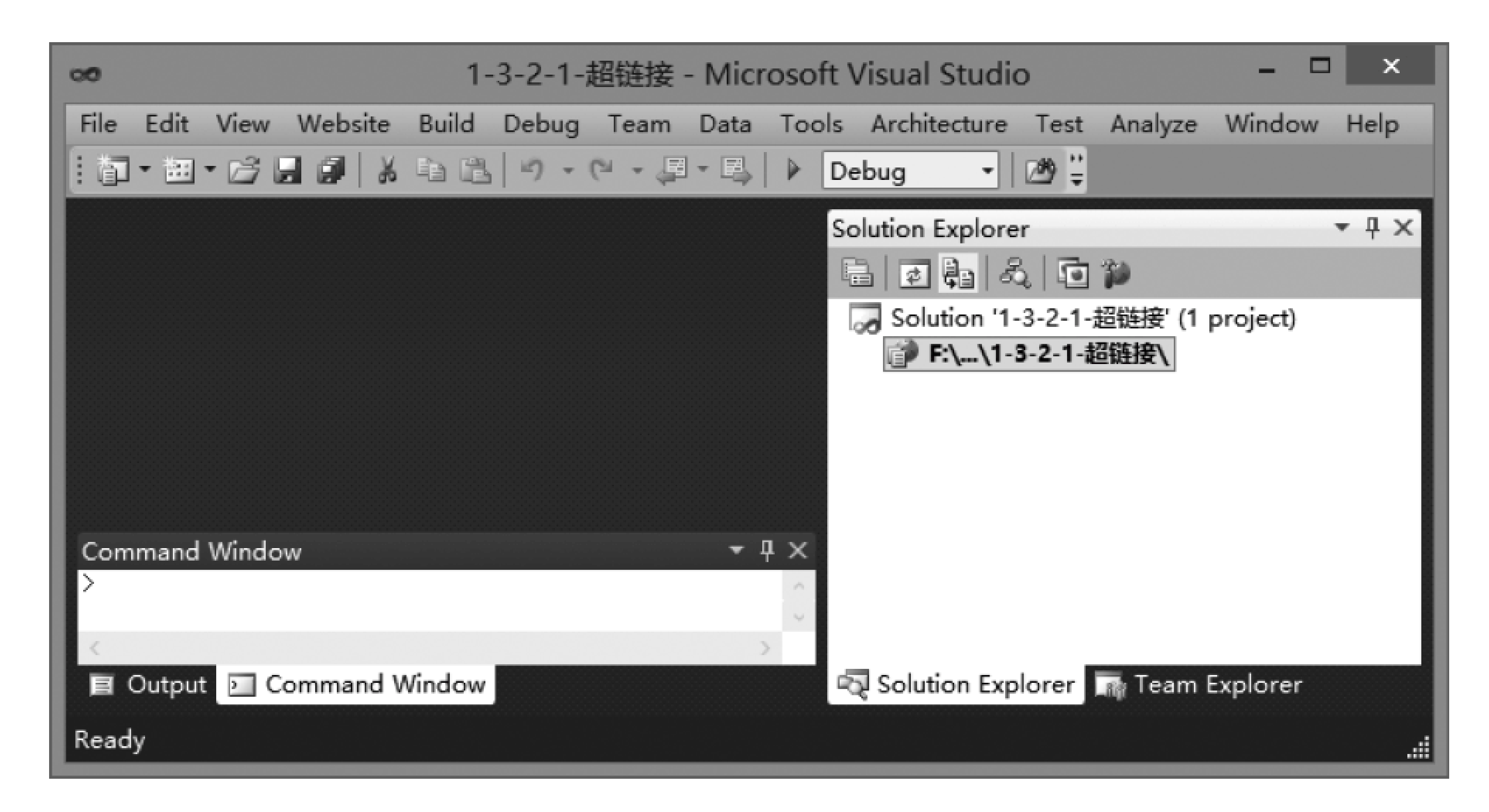


图 1.40　成功打开网站“1-3-2-1-超链接”

（4）添加站内文件夹

鼠标指向网站“1-3-2-1-超链接”，右击弹出如图 1.41 所示的快捷菜单。选择 New Folder，根据需要添加站内文件夹，如图 1.42 所示。

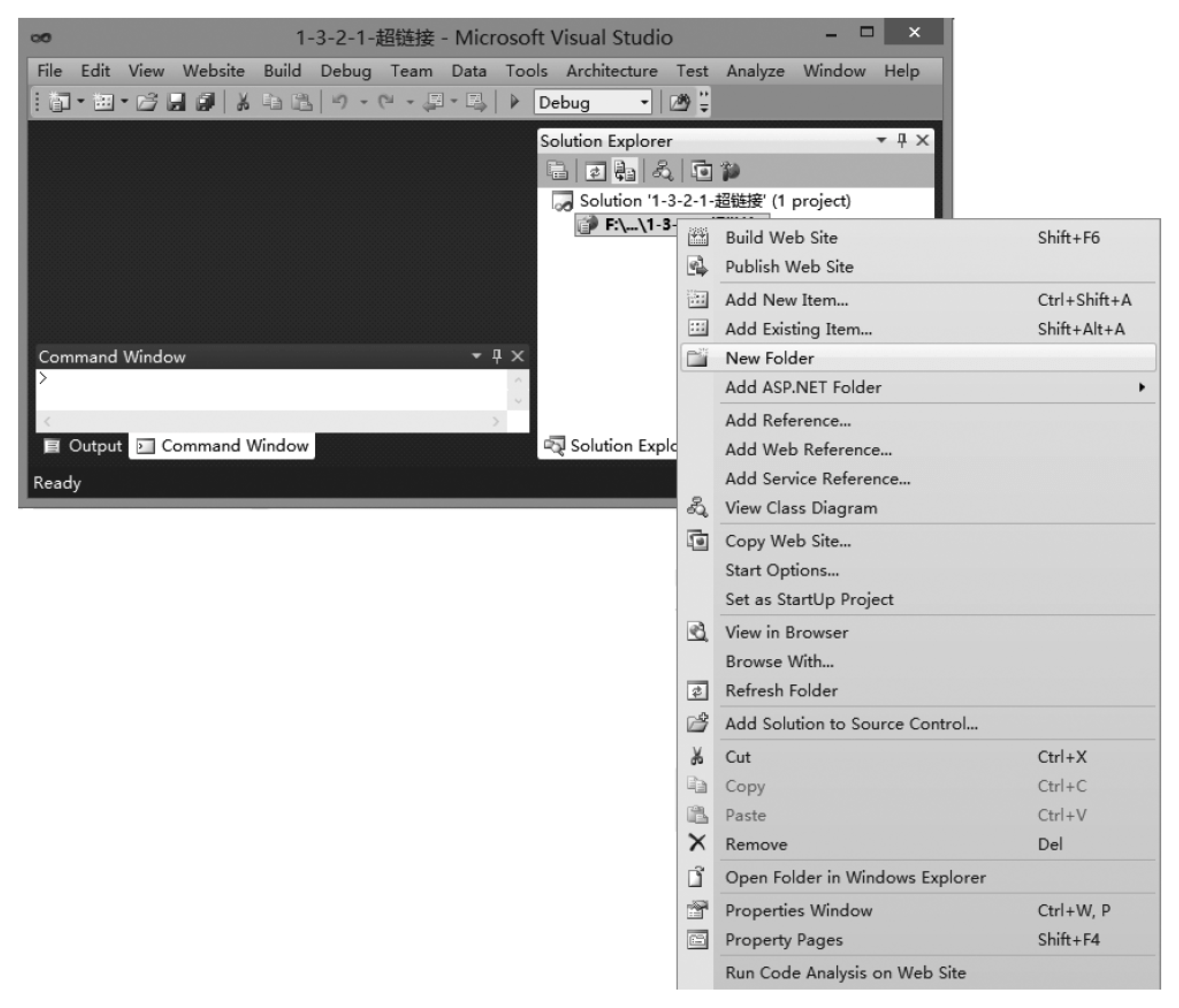


图 1.41　网站快捷菜单

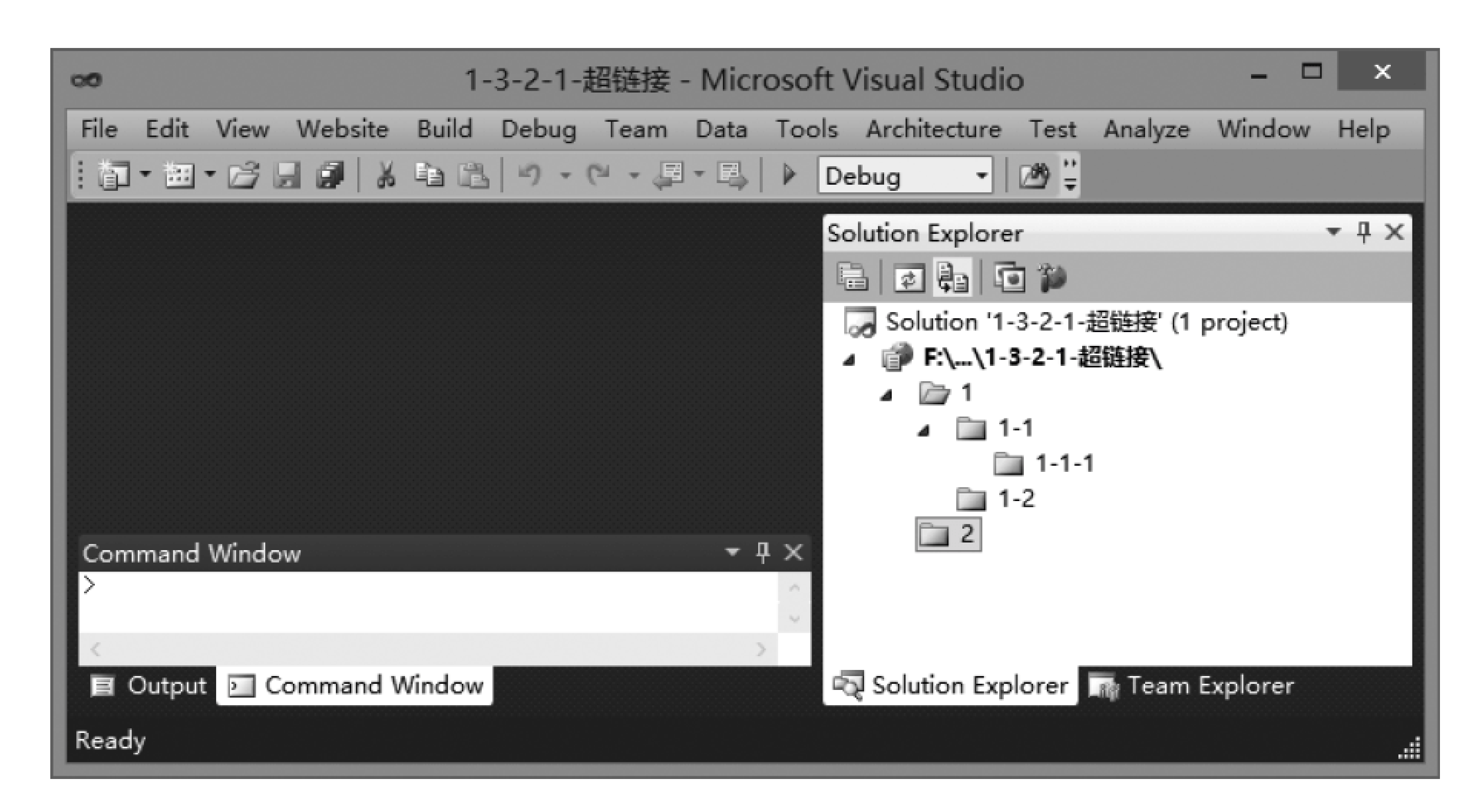


图 1.42　添加文件夹完成后的资源浏览器

(5) 添加网页 Master.htm 和 1-1-1-1-Page.htm

按图 1.43 所示,将 Master.htm 和 1-1-1-1-Page.htm 添加到解决方案中。

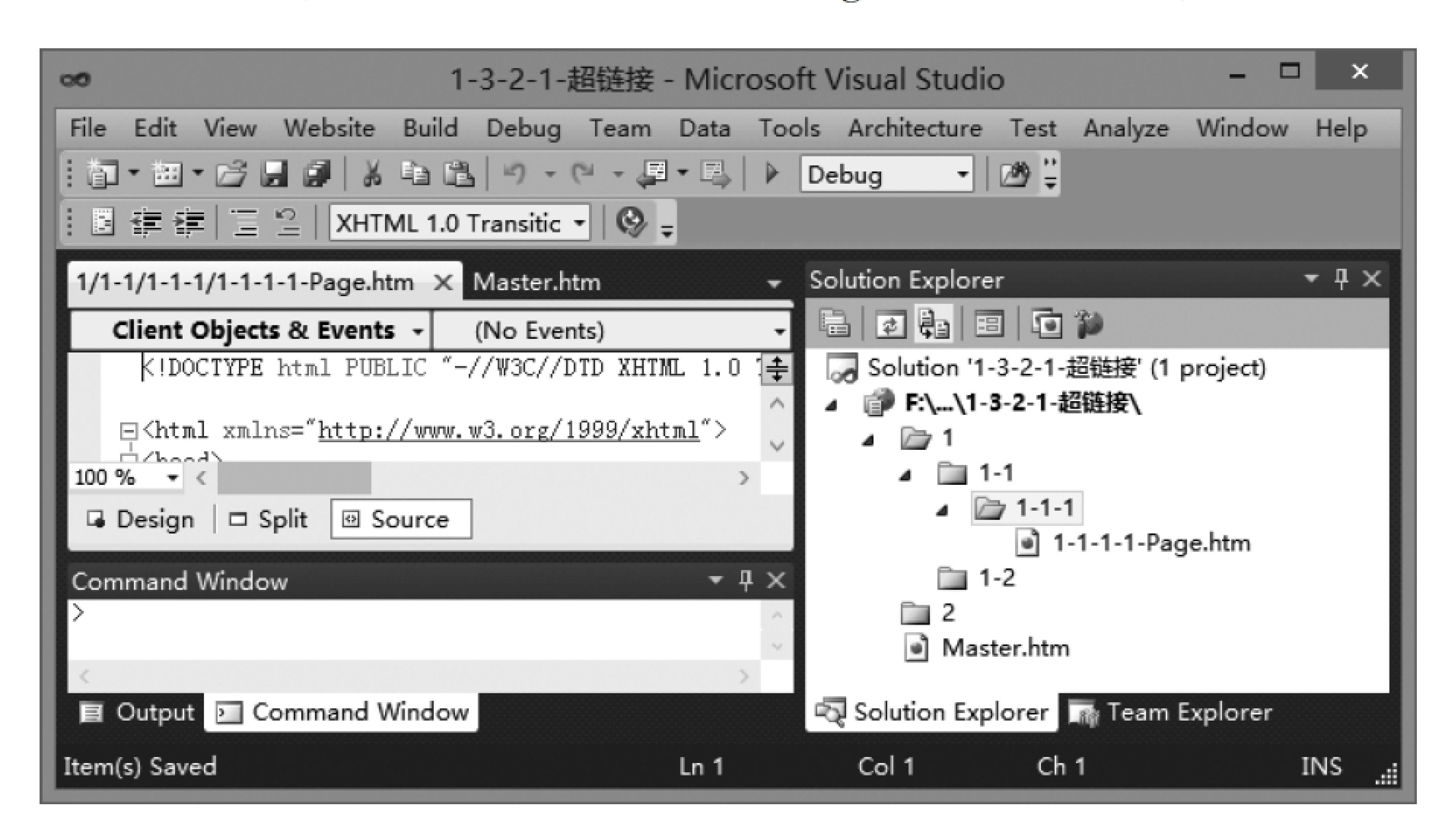


图 1.43　添加两个 htm 类型文件完成后的资源浏览器

(6) 编辑网页 Master.htm

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head><title>实验 1-3-2-1-超链接</title></head>
    <body>
        <p><a href="1/1-1/1-1-1/1-1-1-1-Page.htm">站内链接-不同网页</a>
        </p>
        <p><a href="http://www.w3school.com.cn">站外链接-万维网页面链接</a>
        </p>
        <p><a href="#C4">站内链接-同一网页-查看 Chapter 4。</a></p>
        <h2>Chapter 1</h2><p>This chapter explains ba bla bla</p>
```

```
        <h2>Chapter 2</h2><p>This chapter explains ba bla bla</p>
        <h2>Chapter 3</h2><p>This chapter explains ba bla bla</p>
        <h2><a name="C4">Chapter 4</a></h2><p>This chapter explains ba bla bla</p>
        <h2>Chapter 5</h2><p>This chapter explains ba bla bla</p>
        <h2>Chapter 6</h2><p>This chapter explains ba bla bla</p>
        <h2>Chapter 7</h2><p>This chapter explains ba bla bla</p>
    </body>
</html>
```

(7) 编辑网页 1-1-1-1-Page. htm

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>站内链接-不同网页</title>
    </head>
    <body>
        <a href="../../../Master.htm">返回主页</a>
    </body>
</html>
```

说明：出现.../一次，返回到上级目录一次。例如，当前文件 1-1-1-1-Page. htm，其目录为"1-3-2-1-超链接/1/1-1/1-1-1"。对于"../../../Master. htm"从左向右的第一个"../"，返回到"1-3-2-1-超链接/1/1-1"；第 2 个"../../"，返回到 1-3-2-1-超链接/1；第 3 个"../../../"，返回到"1-3-2-1-超链接"，Mater. htm 在"1-3-2-1-超链接"目录下。

(8) 浏览网页 Master. htm

在如图 1.44 所示的主页面中，单击"站内链接-不同网页"，则打开图 1.45 所示的页面。在如图 1.44 所示的主页面单击"站外链接-万维网页面链接"，则打开图 1.46 所示的页面。在如图 1.44 所示的主页面中单击"站内链接-同一网页-查看 Chapter 4"，则打开图 1.47 所示的页面。



图 1.44 主页面

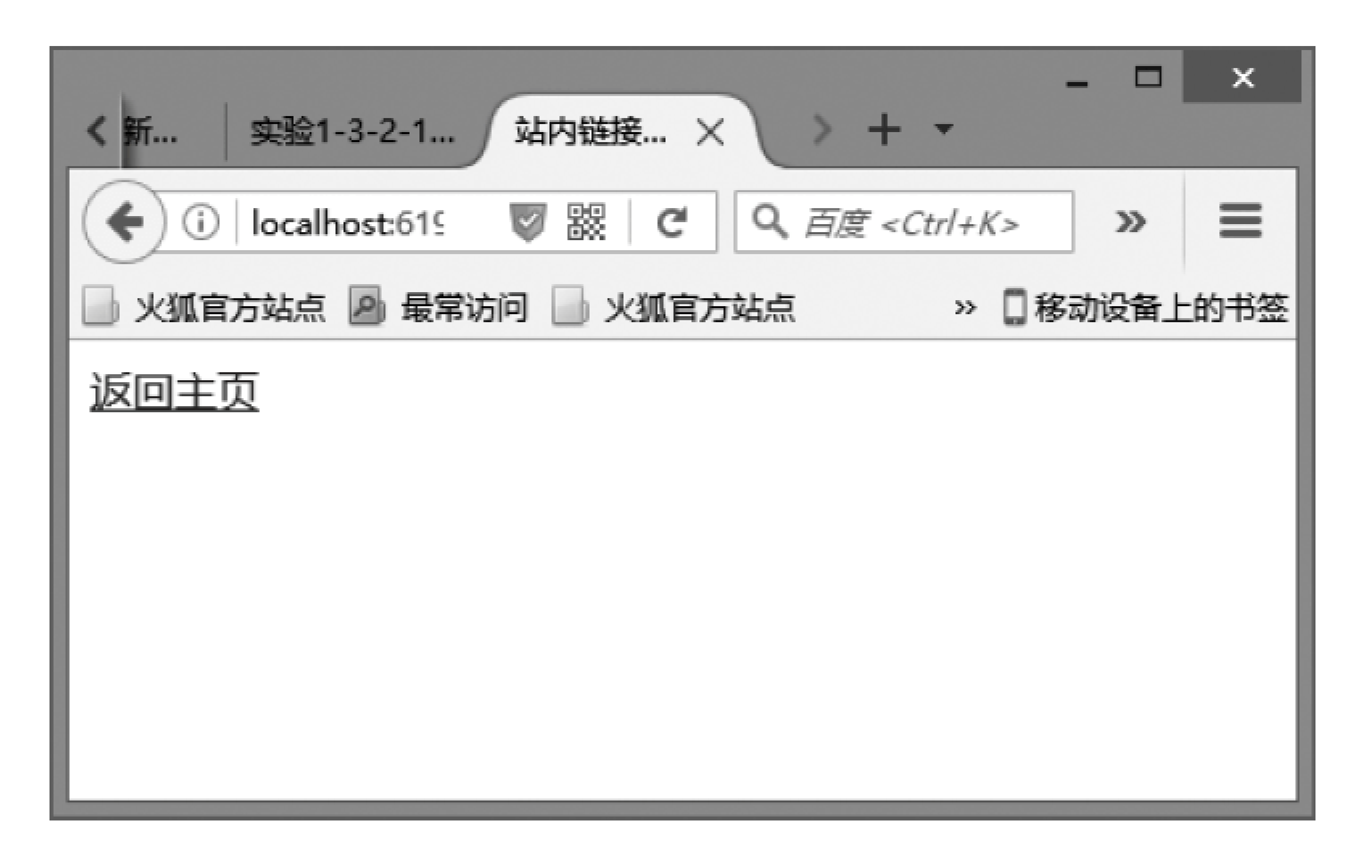


图 1.45　站内链接到不同网页



图 1.46　站外链接到 W3school

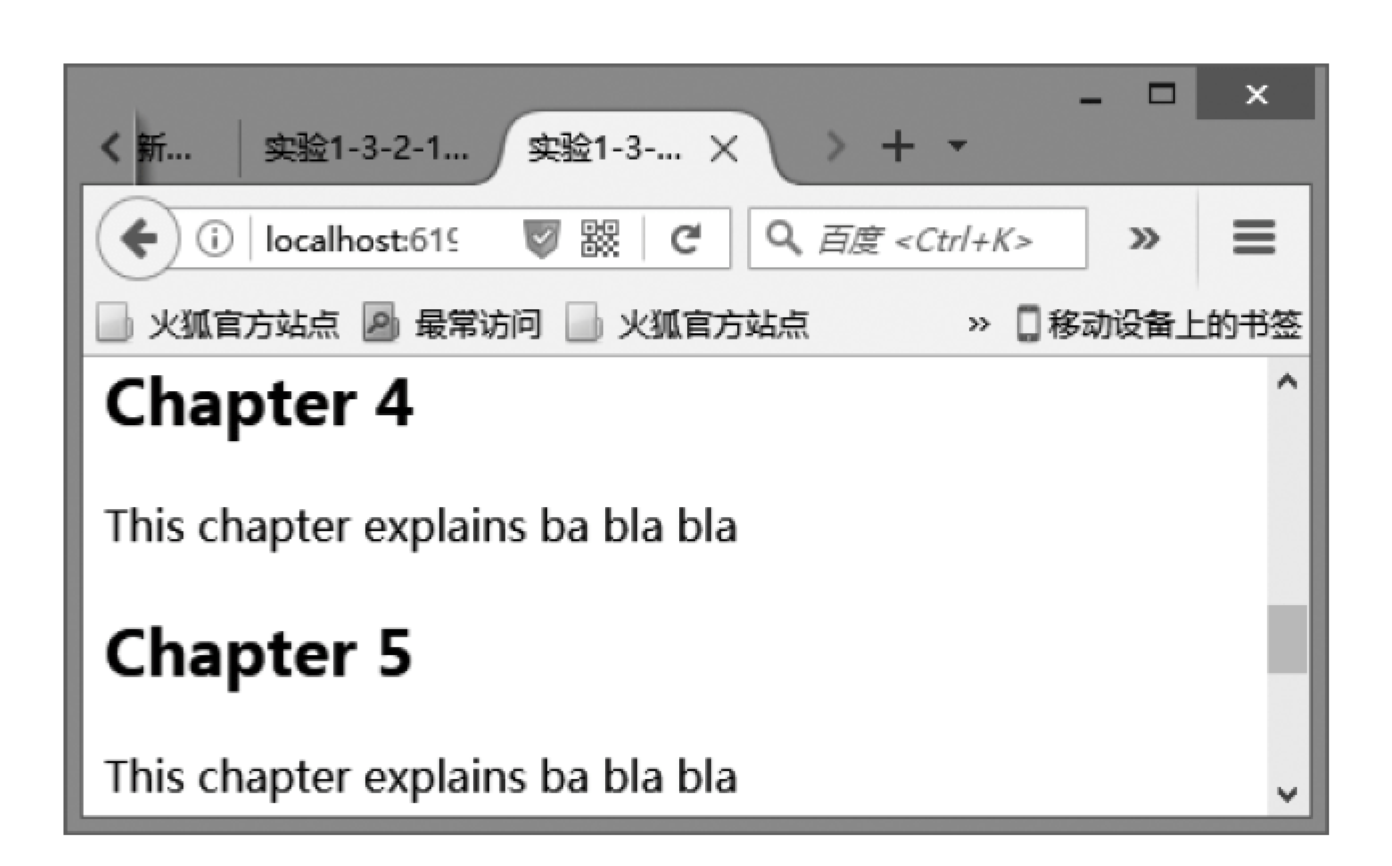


图 1.47　站内链接到同一网页不同元素

2. 任意源的图像

图片源有站内和站外两种情况。站内又分为相同目录和不同目录。相同目录指图像文件与图像标签所在的网页文档处于同一个目录下。不同目录指图像文件与图像标签所在的网页文档分处于不同目录。以图像标签所在的网页文档为参照，不同目录分三

种情况。一是图片文件在某一目录中,但图片文件的某层上级目录与网页文档在同一层目录。二是图片文件在网页文档的某层上级目录中。三是图片文件既不在网页文档的上级目录也不在网页文档本级的某层下级目录。实验所用的网站目录结构如图 1.48 所示。

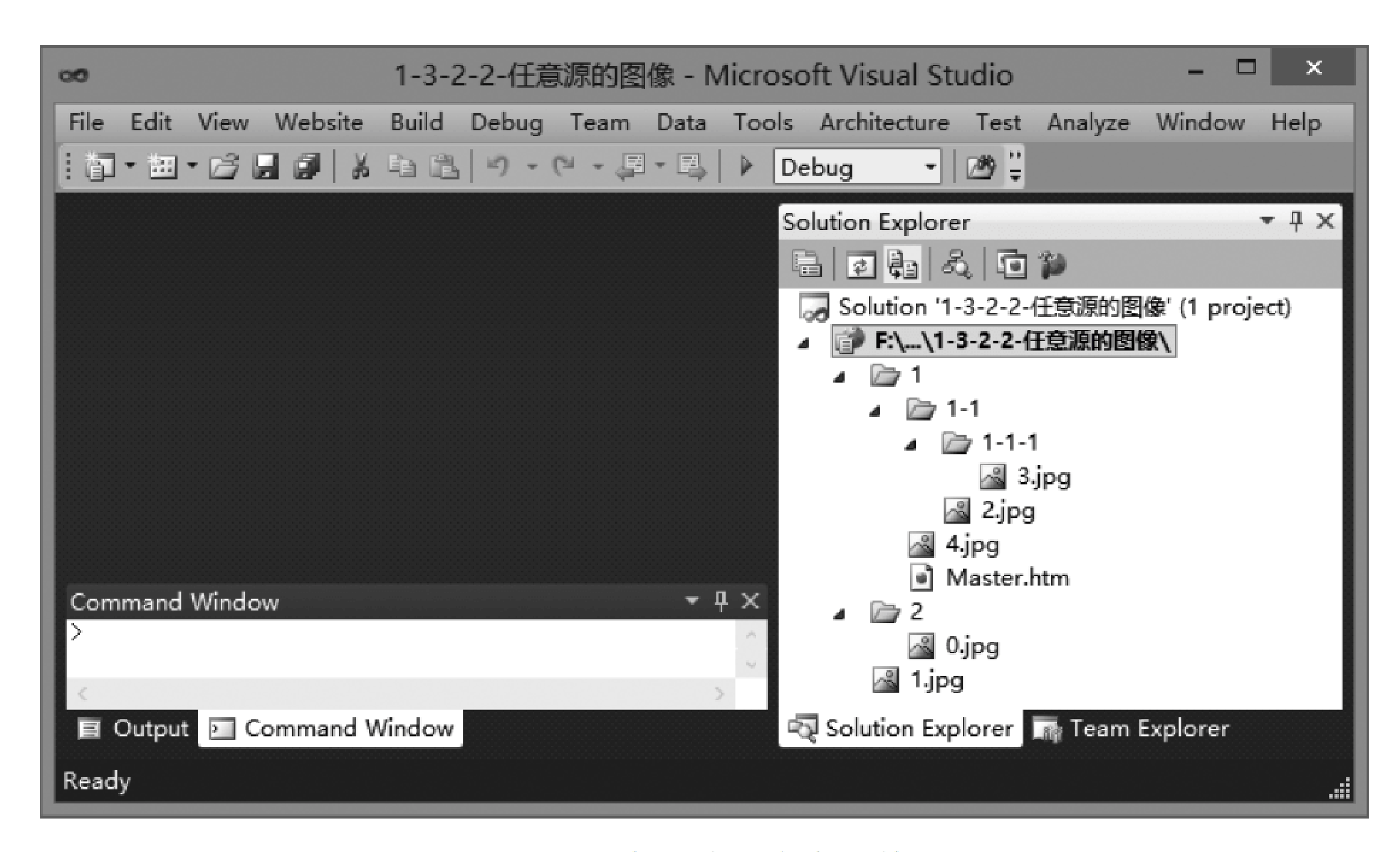


图 1.48 任意源的图像资源管理器

(1) 源程序

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>实验 1-3-2-2-任意源的图像</title>
    </head>
    <body>
        <table>
            <tr>
                <td>站内同级图片<img src="4.jpg" alt=""/></td>
                <td>站内下级图片<img src="1-1/2.jpg" alt=""/></td>
                <td>站内下级图片<img src="1-1/1-1-1/3.jpg" alt=""/></td>
            </tr>
            <tr>
                <td>站内上级图片<img src="../1.jpg" alt=""/></td>
                <td>站内不同级图片<img src="../2/0.jpg" alt=""/></td>
                <td>站外图片<img src="http://www.w3school.com.cn/i/eg_
                buttonnext.gif" alt=""/></td>
            </tr>
        </table>
    </body>
</html>
```

(2) 浏览器显示

浏览器显示结果如图 1.49 所示。



图 1.49 任意源的图像实验浏览器浏览的结果

1.4 实验任务

1. 实验题目

HTML 任意源的图像和任意位置的超链接集成演示。

2. 实现功能

综合利用所学的标签,利用网页展示 1.3 节的内容。提交电子文档,一个学生一个目录,目录名称为"1-4-学号后三位+姓名"。

3. 实验类型

综合设计。

4. 实验要求

利用 Table 整合 1.3.1 和 1.3.2。
使用 H1～H3 标签。
body 要有背景图片。
不同逻辑的 td,运用不同的背景色或图片。
至少 3 个 HTML 文档,1 个索引文档或主文档,另 2 个为链接和图像。

5. 实验环境

(1) 计算机：PC、内存 8GB、主频 1.8GHz 及以上、硬盘 500GB 及以上。
(2) 操作系统：Windows XP、Windows 7、Windows 8、Windows 10。
(3) 开发环境：Visual Studio 2010 或 Adobe Dreamweaver。
(4) 浏览器：IE8 及以上、Chrome、Firefox、Safari、Edge、QQ 浏览器等。

6. 实验原理

(1) 给出不同位置的 URL。
(2) 给出站内目录结构。

7. 源代码

(1) 辅以必要的注释。
(2) 错落有致，结构清晰，易读性强。

8. 遇到的问题及解决办法

(1) 字数 300 字以上。
(2) 语言简洁扼要，条理清楚。

第 2 章 chapter 2

表单与 CSS

■ 知识目标

- 掌握表单的基本属性及其与 Web 服务器的交互方法
- 掌握 INPUT 标签 9 种类型的用法
- 理解数据、结构和样式分离的原因
- 掌握 CSS 的 3 种用法
- 掌握 CSS 选择器的用法
- 掌握 CSS 的定位方式

■ 能力目标

- 能够根据实际需求恰当设置表单属性
- 能够根据实际应用准确选用 INPUT 类型
- 能够应用外部样式分离结构和样式
- 能够运用 CSS 灵活设置标签的样式
- 恰当运用定位设计界面

■ 素质目标

- 设计精美、精致的网页
- 从通信的视角看待浏览器与服务器
- 结构、样式、行为与内容的良好分离

■ 教学重点

- 表单的基本属性(id、action、method、runat)
- input 的 9 种类型和基本属性(id、type、name、value、size、maxlength)
- 表单与 ASP. NET 服务器进行交互
- 样式的基本用法、嵌套、重复与冲突
- 盒子模型
- 菜单

■ 教学难点

- 识别浏览器端程序和服务器端程序的运行错误

■ **建议学时**

- 理论：6 学时
- 实验：6 学时

2.1 表　　单

HTML 表单用于搜集不同类型的用户输入，将搜集到的数据和动作请求传给服务器，服务器根据不同的数据和请求进行不同的处理，保存处理结果并向浏览器发回响应。它是浏览器和服务器进行交互的桥梁。

<form>元素定义 HTML 表单，不在浏览器中显示。按表单元素的填写方式将表单分为输入类控件和列表类控件。输入类控件一般以 input 标记开始，说明这一表单元素需要用户的输入；列表类以 select 开始，表示用户需要选择。

1. <input>元素

<input>元素是最重要的表单元素，服务器根据 name 属性来识别一个表单内的<input>元素。<input>元素有很多形态，包括文本框、密码框、复选框、单选按钮、提交按钮等，由 type 属性的实际枚举值确定，如表 2.1 所示。

表 2.1　type 枚举值说明

type 值	说　明	type 值	说　明
text	文本字段	submit	提交按钮
password	密码域，用户在输入时不显示具体内容，以 * 代替	reset	重置按钮
radio	单选按钮	hidden	隐藏域
checkbox	复选框	file	文件域
button	普通按钮		

size 属性规定以字符数计的输入字段宽度。maxlength 属性规定输入字段的最大长度，以字符个数计。value 属性为 input 元素设定值。对于不同的输入类型，value 属性的用法也不同，具体如下：

① type="button""reset""submit"：定义按钮上的显示的文本。

② type="text""password""hidden"：定义输入字段的初始值。

③ type="checkbox""radio""image"：定义与输入相关联的值。

<input type="checkbox">和<input type="radio">中必须设置 value 属性。而 value 属性无法与<input type="file"> 一同使用。

(1) 文本框

文本框只能输入单行文本，如需要输入多行，则需要使用 textarea 元素。

例 2-1-1-1　一个文本框应用示例。

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>例 2-1-INPUT-TEXT</title>
    </head>
    <body>
        身份证号<input type="text" name="PID" value="" size="20" maxlength="18"/>
    </body>
</html>
```

浏览器显示如图 2.1 所示。

身份证号

图 2.1　例 2-1-1-1 的浏览器显示

(2) 密码域

例 2-1-1-2　一个密码域应用示例。

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>例 2-2-INPUT-PASSWORD</title>
    </head>
    <body>
        密码<input type="password" name="PWD" value="" size="30" maxlength=
        "18"/>
    </body>
</html>
```

在浏览器的密码域中输入"1234567",显示如图 2.2 所示。

(3) 单选

例 2-1-1-3　一个单选框应用示例。

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>例 2-3-INPUT-RADIO</title>
    </head>
    <body>
        <pre>性 别:<input type="radio" name="sex" value="M" checked=
        "checked" />男<input type="radio" name="sex" value="F"/>女</pre>
    </body>
</html>
```

浏览器显示如图 2.3 所示。

密码●●●●●●●

图 2.2　例 2-1-1-2 的浏览器显示

性　别: ◉男 ○女

图 2.3　例 2-1-1-3 的浏览器显示

说明：默认值选中"男"由 checked 属性设置，可根据实际改变选择。而且各个单选按钮的 name 属性值一定要相同。

(4) 复选

例 2-1-1-4 一个复选框应用示例。

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>例 2-4-INPUT-CHECKBOX</title>
    </head>
    <body>
        <p>请选择你的爱好:</p>
        <p>
        <input type="checkbox" name="test1" value="A1" checked="checked"/>上网
        <input type="checkbox" name="text2" value="A2"/>游泳
        <input type="checkbox" name="test3" value="A3"/>登山
        <input type="checkbox" name="test4" value="A4"/>写作
        </p>
    </body>
</html>
```

请选择你的爱好：

☑上网 ☐游泳 ☐登山 ☐写作

图 2.4 例 2-1-1-4 的浏览器显示

浏览器显示如图 2.4 所示。

说明：默认选项“上网”由 checked 属性设置，可根据实际改变选择。而且各个复选按钮的 name 属性值需要各不相同。

(5) 按钮

按钮有 3 类，即提交按钮、普通按钮和重置按钮，一个表单中有且仅有一个提交按钮，用于触发表单动作。重置按钮用于清除表单中的所有数据。普通按钮在一个表单中可有可无、可多可少，根据实际选择。普通按钮须用脚本方法名设置 onclick 属性值。

例 2-1-1-5 一个按钮应用示例。

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>例 2-5-INPUT-按钮</title>
    </head>
    <body>
        <h5>三类按钮:</h5>
        <p>
            <input name="button1" type="submit" value="提交"/>
            <input name="button2" type="reset" value="重置"/>
            <input name="button3" type="button" value="普通按钮"/>
        </p>
    </body>
</html>
```

浏览器显示如图 2.5 所示。

三类按钮：

[提交] [重置] [普通按钮]

图 2.5 例 2-1-1-5 的浏览器显示

(6) 隐藏域

隐藏域主要用来传递一些参数,而这些参数不需要在页面中显示。当浏览者提交表单时,隐藏域的内容会一起提交给处理程序。

例 2-1-1-6　一个隐藏域应用示例。

```
<html>
    <head><title>例 2-6-隐藏域和 action 属性对比示例</title></head>
    <body>
        <form name="exam5" action="例 2-06-INPUT-隐藏域的 exam1.html"
            method="get">
        下面是几种不同属性的文本字段:
        <p>姓名:<input type="text" name="username" size="15" /></p>
        <p>年龄:<input type="text" name="age" size="15" maxlength="3"/></p>
        <p><input type="hidden" name="page_id" value="example"/>
        <input type="submit" name="Submit" value="提交"/></p>
        </form>
    </body>
</html>
```

浏览器显示如图 2.6 所示。

(7) 文件域

文件域用于浏览选择需要上传的文件。

例 2-1-1-7　一个文件域应用示例。

```
<html>
    <head><title>例 2-7-文件域</title></head>
    <body>
        <form action="mailto:fast@126.com" name="research" method="post">
            下面是某网站的注册页面:
            <p>用户名:<input name="username" type="text" size="20"/></p>
            <p>密码:
                <input name="password1" type="password" size="20"/>
            </p>
            <p>请上传你的头像:<input type="file" name="picture"/></p>
        </form>
    </body>
</html>
```

浏览器显示如图 2.7 所示。

下面是几种不同属性的文本字段:

姓名:

年龄:

提交

图 2.6　例 2-1-1-6 的浏览器显示

下面是某网站的注册页面:

用 户 名:

密　　码:

请上传你的头像: 浏览... 未选择文件。

图 2.7　例 2-1-1-7 的浏览器显示

单击“浏览”按钮，打开“文件上传”对话框，如图 2.8 所示。

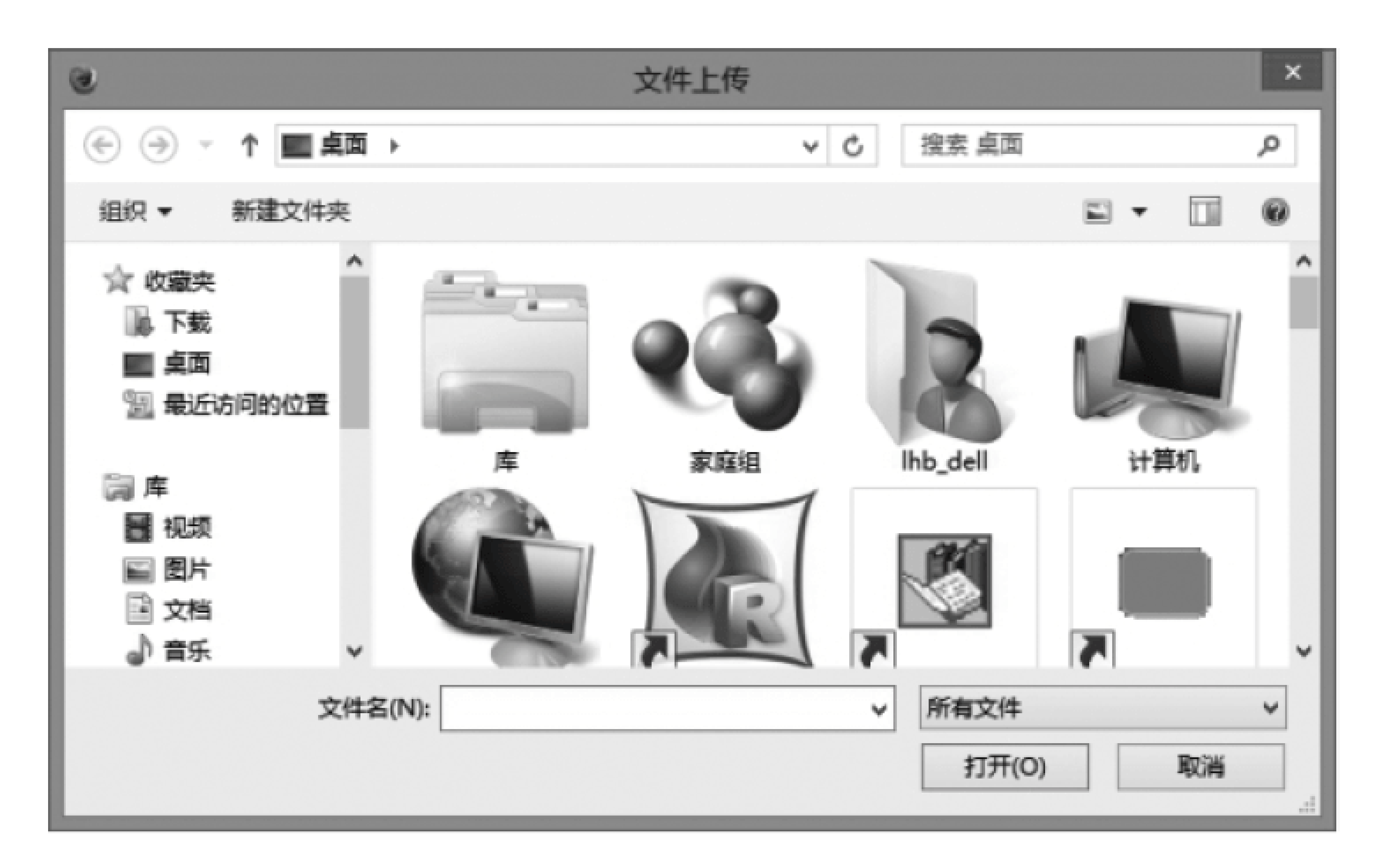


图 2.8 “文件上传”对话框

在图 2.8 中选中一个文件，浏览器显示如图 2.9 所示。

下面是某网站的注册页面：

用 户 名:

密 码:

请上传你的头像： 浏览... 02-3.gif

图 2.9 例 2-1-1-7 的浏览器显示

2. 列表元素

所有主流浏览器都支持<select>标签。select 元素可创建单选或多选菜单。<select>元素中的<option>标签用于定义列表中的可用选项。

例 2-1-1-8 一个列表应用示例。

```
<html>
    <head><title>例 2-8-select</title></head>
    <body>
        <p>证件类型
            <select name="cardtype">
                <option value="id_card">身份证</option>
                <option value="stu_card">学生证</option>
                <option value="drive_card">驾驶证</option>
                <option value="other_card">其他证件</option>
            </select>
        </p>
        <p>关心的栏目
            <select name="content" size="3" multiple="multiple">
                <option value="m1">体育栏目</option>
                <option value="m2">科技栏目</option>
                <option value="m3">新闻栏目</option>
                <option value="m4">汽车栏目</option>
                <option value="m5">房产栏目</option>
            </select>
```

```
            </p>
        </body>
</html>
```

浏览器显示如图 2.10 所示。

说明：下拉列表中的 size 属性值指明列表中显示选项数，multiple 指明多选，默认为单选。当 select 元素中给出 multiple 属性时，按住 Ctrl 键进行多选，一个选中 2 个栏目的浏览器显示如图 2.11 所示。



图 2.10　例 2-1-1-8 的浏览器显示　　**图 2.11　例 2-1-1-8 中下拉列表多选的浏览器显示**

3. textarea 元素

<textarea> 标签定义多行的文本输入控件。文本区中可容纳无限数量的文本，其中文本的默认字体是等宽字体（通常是 Courier）。可以通过 cols 和 rows 属性来规定 textarea 的尺寸，不过更好的办法是使用 CSS 的 height 和 width 属性。文本输入区内的文本行间用 "%OD%OA"（回车/换行）进行分隔。使用<textarea>标签的 wrap 属性设置文本输入区内的换行模式。

例 2-1-1-9　一个 textarea 元素应用示例。

```
<html>
    <head><title>例 2-9-textarea</title></head>
    <body>
        您的意见对我很重要
          <textarea name="info" cols="35" rows="7">请将意见输入此区域
              </textarea>
    </body>
</html>
```

浏览器显示如图 2.12 所示。

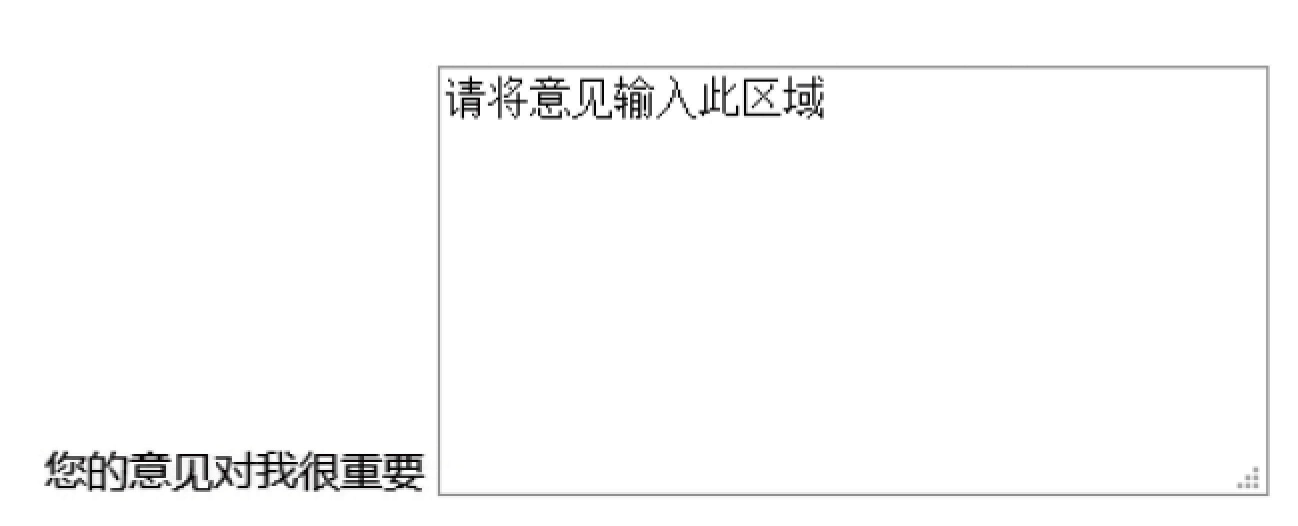


图 2.12　例 2-1-1-9 的浏览器显示

4. form 表单与 ASP.NET 服务器互动

表单用于向服务器提交处理请求。用户填写完信息后做提交操作，将表单内容从客户端的浏览器传送到服务器上，经过服务器处理程序后，再将用户所需信息送回客户端的浏览器上，这样网页就具有了交互性。action 属性定义提交表单时执行的动作。向服务器提交表单的通常做法是使用提交按钮。通常表单会被提交到 Web 服务器的网页中。

表单和 Web 服务器的 HTTP 通信方式由 method 属性设置，有 get 和 post 两种方法，两种方法的区别如下。

(1) 传输格式

GET 请求，请求的数据会附加在 URL 之后，以?分割 URL 和传输数据，多个参数用 & 连接。URL 的编码格式采用 ASCII 编码，而不是 unicode，即所有的非 ASCII 字符都要编码之后再传输。

POST 请求会把请求的数据放置在 HTTP 请求包的包体中。上面的 item=bandsaw 就是实际的传输数据。因此，GET 请求的数据会暴露在地址栏中，而 POST 请求则不会。

(2) 传输数据规模

HTTP 规范中没有对 URL 的长度和传输的数据大小进行限制。但是在实际开发过程中，对于 GET，特定的浏览器和服务器对 URL 的长度有限制。因此，在使用 GET 请求时，传输数据会受到 URL 长度的限制。

对于 POST，由于不是 URL 传值，理论上是不会受限制的，但实际上各个服务器对 POST 提交数据大小是有限制的，Apache、IIS 都有各自的配置。

(3) 安全性

POST 的安全性比 GET 的高。这里的安全是指真正的安全，而不同于上面 GET 提到的安全方法中的安全，上面提到的安全仅仅是不修改服务器的数据。比如，在进行登录操作时，通过 GET 请求，用户名和密码都会暴露在 URL 上，因为登录页面有可能被浏览器缓存以及其他人查看浏览器的历史记录的原因，此时的用户名和密码就很容易被他人拿到。除此之外，GET 请求提交的数据还可能造成跨站请求伪造攻击。

(4) 运行协议

HTTP 中的 GET、POST、SOAP 协议都是在 HTTP 上运行。

表单间不能嵌套使用。表单能够包含 input 元素，比如文本字段、复选框、单选框、提交按钮等，还可以包含 select、menus、textarea、fieldset、legend 和 label 元素。form 元素是块级元素，前后会产生折行。所有浏览器都支持 form 标签。

如果表单中的 input、label、select、textarea、fieldset、legend 元素需要被服务器端的后台 C# 或 VB 应用程序处理，不但表单元素需要给出 runat 属性值为 server，而且被读写的 input、label、select、textarea、fieldset、legend 元素也需要给出 runat 属性值为 server。同时，所有需要被服务器处理的元素必须给出一个唯一的 id 属性值。

get 通信方式的实例将在 DOM 部分中讲述，post 通信方式的具体实例如例 2-1-1-10

所示。

例 2-1-1-10 post 通信方式实例。

(1) 例 2-1-1-10-form-fieldset.aspx

```
<%@Page Language="C#" AutoEventWireup="true" CodeFile="例 2-1-1-10-form-fieldset.
aspx.cs" Inherits="理论_例 2_1_1_10_form_fieldset" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <fieldset>
                <legend>健康信息</legend>
                身高:<input id='height' type="text"  runat="server"
                value='170'/>
                体重:<input id='weight' type="text"  runat="server"
                value='65'/>
            </fieldset>
            <p><input type="submit" value='提交'/></p>
            <label id='inf' runat="server"></label>
        </div>
    </form>
</body>
</html>
```

(2) 例 2-1-1-10-form-fieldset.aspx.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
public partial class 理论_例 2_1_1_10_form_fieldset : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        this.inf.InnerText="身高="+this.height.Value+"cm,"+"体重="+this.weight.
        Value+"Kg";
    }
}
```

(3) 浏览器显示

浏览器显示如图 2.13 所示。

健康信息

身高：170 体重：65

提交

身高=170cm,体重=65Kg

图 2.13 例 2-1-1-10 的浏览器显示

说明，默认身高和体重分别为 170cm 和 65kg，由 input 的 value 属性设置。用户在浏览器端修改后，单击提交按钮，则服务器端刷新网页。此外，对于.aspx 格式文件，form 的默认通信方式为 post。如果 form 没指明 action，默认的 form 会自动设置 action 值。按 F12 键，进入开发人员工具界面，如图 2.14 所示。



图 2.14 火狐开发人员工具查看器下的文档

2.2 层叠样式表-CSS

HTML 标签原本被设计为用于定义文档内容。通过使用<h1><p><table>这样的标签，HTML 的初衷是表达“这是标题”“这是段落”“这是表格”之类的信息。同时文档布局由浏览器来完成，而不使用任何的格式化标签。

由于 Netscape 和 Internet Explorer 两种主要的浏览器不断地将新的 HTML 标签和属性(比如字体标签和颜色属性)添加到 HTML 规范中，创建文档内容清晰地独立于文档表现层的站点变得越来越困难。标签能够实现的样式控制较为单一，单一的网页控制方式要求程序员本身的工作量大量增加，而专业的美工无法在网页设计阶段充分发挥所长，浪费了大量人力、时间成本。当页面需要改变样式时，重复大量的工作严重降低了程序开发人员的开发效率。随着网页设计技术的发展，人们已经不满足原有的一些 HTML 标记，而是希望能够为页面内容添加一些更加绚丽的属性，如鼠标标记、渐变效果等。

为了解决这个问题，万维网联盟(W3C)在 HTML 4.0 之外创造出样式(Style)。CSS 技术的发展使这些成为现实。CSS 是 Cascading Style Sheet 的缩写，可以翻译为“层叠样式表”或“级联样式表”，即“样式表”。把样式添加到 HTML 4.0 中，解决了内容与

表现分离的问题。所有的主流浏览器均支持层叠样式表。

样式通常存储在样式表中。外部样式表可以极大地提高工作效率，它通常存储在CSS文件中。仅仅编辑一个简单的CSS文档，外部样式表就能同时改变站点中所有页面的布局和外观。由于允许同时控制多重页面的样式和布局，CSS可以称得上是Web设计领域的一个突破。网站开发者能够为每个HTML元素定义样式，并根据需要将之应用于任意多的页面中。如需全局更新，只需简单地改变样式，然后网站中的所有元素均会自动地更新。

多个样式定义可层叠为一。样式表允许以多种方式规定样式信息。样式可以规定在单个的HTML元素中，如在HTML页的头元素中，或在一个外部的CSS文件中。甚至可以在同一个HTML文档内部引用多个外部样式表。

当同一个HTML元素被不止一个样式定义时，会使用哪个样式呢？

一般而言，所有的样式会根据下面的规则层叠于一个新的虚拟样式表中，其中数字大小表示拥有优先权的高低。

① 浏览器缺省设置。

② 外部样式表。

③ 内嵌样式表（位于<head>标签内部）。

④ 行内样式（在HTML元素内部）。

因此，行内样式（在HTML元素内部）拥有最高的优先权，这意味着它将优先于几个样式声明：<head>标签中的样式声明，外部样式表中的样式声明，或者浏览器中的样式声明（默认值）。

1. CSS的优势

① 样式表为页面提供了丰富的美观元素，使页面更加美观和灵活。

② 样式表可以实现内容与样式的分离，方便团队协作开发，大大提高了网站开发效率。

③ 程序员与美工分工非常明确，大大减少了程序员的工作量。

④ 为网站开发提供了新的模式。

2. CSS的功能

① 灵活控制网页中文字的字体、颜色、大小、间距、风格及位置。

② 随意设置一个文本块的行高、缩进，并可以为其加入三维效果的边框。

③ 更方便地为网页中的任何元素设置不同的背景颜色和背景图片。

④ 精确控制网页中各元素的位置。

⑤ 可以与脚本语言相结合，使网页中的元素产生各种动态效果。

从CSS作用的区域分为行内样式、内嵌样式和外部样式。行内样式由HTML标签的Style属性设置，第1章已经介绍，行内样式作用于所在单个标签。内嵌样式作用于所在的单个HTML文件。外部样式作用于所导入的任何HTML文档。

2.2.1 内嵌样式

内嵌样式置于 head 元素里，以<style type="text/css"></style>来书写。<style>标签用于为 HTML 定义样式。其中，type 属性是必需的，定义 Style 元素的内容，其唯一值为 text/css。style 内包含选择器。选择器有标签选择器、类选择器、ID 选择器、子代选择器、包含(后代)选择器、通用选择器以及最后的伪类选择符和分组选择符。

style 元素的内容由 CSS 规则构成。CSS 规则由两个主要的部分构成：选择器以及样式声明。

```
selector {declaration1; declaration2; ... declarationN }
```

选择器通常是需要改变样式的一个或一批 HTML 元素。声明置于花括号内，进行一条或多条样式的声明。每条声明由一个属性和一个值组成。每个属性有一个值，属性和值用冒号分开。声明之间用分号分隔。

```
selector { property1: value1; property2: value2; …propertyN: valueN;}
```

下面这行代码的作用是将 h1 元素内的文字颜色定义为红色，同时将字体大小设置为 14 像素。在这个例子中，h1 是选择器，color 和 font-size 是属性，red 和 14px 是值。

```
h1 {color:red; font-size:14px;}
```

(1) 颜色值

颜色值除了使用英文单词，还可以使用十六进制的颜色值，可访问 http://www.w3school.com.cn/tags/html_ref_colornames.asp 查阅颜色名值对照表。

(2) 多单词值

如果值为若干单词，则要给值加引号，例如 p {font-family: "sans serif";}。

(3) 空格与大小写

是否包含空格不会影响 CSS 在浏览器的工作效果，同样，与 XHTML 不同，CSS 对大小写不敏感。不过存在一个例外：如果涉及与 HTML 文档一起工作，class 和 id 名称对大小写是敏感的。

(4) 选择器分组

分组的选择器就可以分享相同的声明。用逗号就能将需要分组的选择器分开。

例如：h1,h2,h3,h4,h5,h6 { color: green; }。

(5) 子代选择器

语法：

```
父代名>子代名{属性名:属性值;}
```

例 2-2-1-1 子代选择器应用示例 1。

```
<html>
    <head>
        <title>例 2-11-CSS-多层级-父子</title>
```

```
        <style type="text/css">
            div>div>div>div>div{border:1px dashed Green;}
            div{margin-left:10px;margin-top:10px;border:1px solid red;}
        </style>
    </head>
    <body>
      <div style="width:130px;height:130px;">
            <div style="width:110px;height:110px;">
                  <div style="width:90px;height:90px;">
                        <div style="width:70px;height:70px;">
                        <div style="width:50px;height:50px;">
                              <div style="width:30px;height:30px;">
                              <div style="width:10px;height:10px;">
                              </div>
                          </div>
                      </div>
                </div>
          </div>
      </div>
    </body>
</html>
```

浏览器显示如图 2.15 所示。

说明：div＞div＞div＞div＞div｛border：1px dashed green；｝是严格的父子关系，选择有连续 5 对相邻 div 之间的直接父子关系的第 5 个 div。满足连续 5 级父子 div 关系选择条件的有最内层相邻的 3 个 div，因而显示线型为虚线，颜色为绿色。其他不满足连续 5 级 div 父子关系选择条件的 div 线型为实线，颜色为红色。父代名和子代名可以标签选择器、类选择器、ID 选择器、属性选择器、属性值选择器、标签属性选择器、标签属性值选择器。

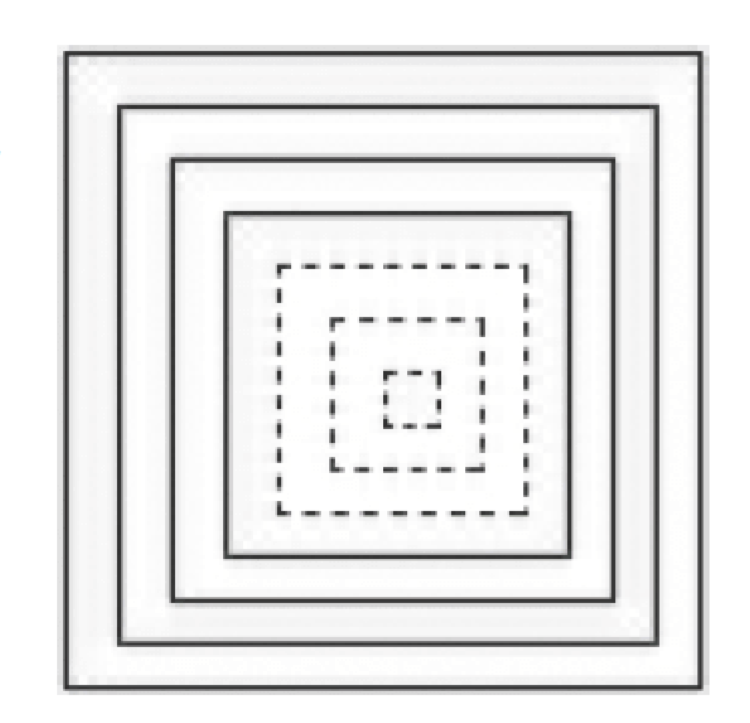


图 2.15　例 2-2-1-1 的浏览器显示

例 2-2-1-2　子代选择器应用示例 2。

```
<html>
    <head>
        <title>例 2-12-CSS-多层级-父子</title>
        <style type="text/css">
            div>div>div>div>div{border:1px dashed Green;}
            div{margin-left:10px;margin-top:10px;border:1px solid red;}
        </style>
    </head>
    <body>
        <div style="width:130px;height:130px;">
```

```
                <div style="width:110px;height:110px;">
                    <div style="width:90px;height:90px;">
                        <div style="width:70px;height:70px;">
                            <div style="width:50px;height:50px;">
                                <form action="">
                                    <div style="width:30px;height:30px;">
                                        <div style="width:10px;height:10px;">
                                        </div>
                                    </div>
                                </form>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </body>
</html>
```

浏览器显示如图 2.16 所示。

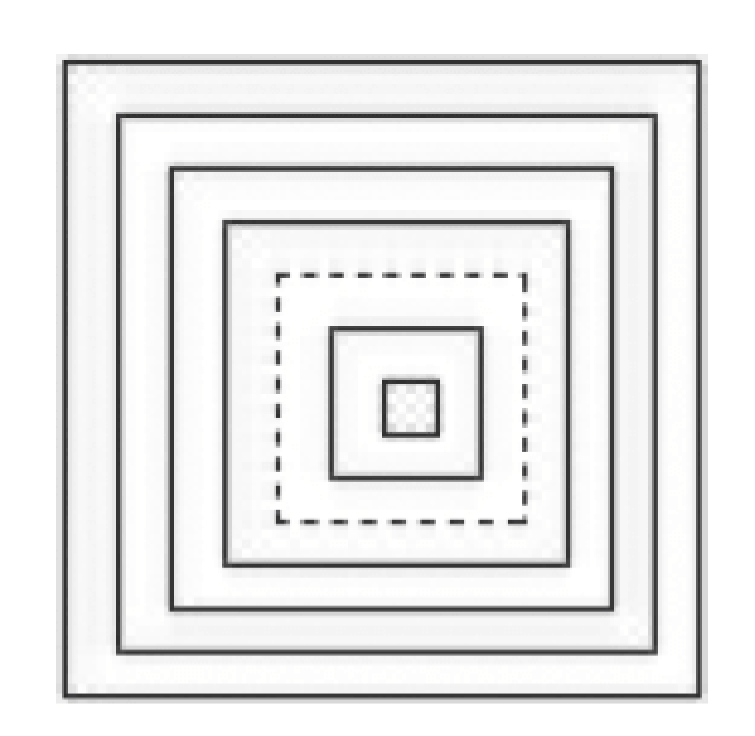


图 2.16　例 2-2-1-2 的浏览器显示

说明：例 2-2-1-2 与例 2-2-1-1 的区别是在第 5 个和第 6 个 div 之间加了一个 form 标签，这样满足连续 5 对相邻 div 父子关系的最内层 div 只有一个，其他均不满足。因此，只有第 5 个 div 线型为虚线，颜色为绿色，其他 div 线型为实线，颜色为红色。

（6）后代选择器

语法：

父代名>后代名{属性名：属性值；}

例如：div div div div div div { color：green；}有 5 对相邻 div 之间的父代与后代关系。

例 2-2-1-3　后代选择器应用示例 1。

```
<html>
    <head>
        <title>例 2-13-CSS-多层级-包含</title>
        <style type="text/css">
            div div div div div{border:1px dashed Green;}
            div{margin-left:10px;margin-top:10px;border:1px solid red;}
        </style>
    </head>
    <body>
      <div style="width:130px;height:130px;">
          <div style="width:110px;height:110px;">
```

```
                <div style="width:90px;height:90px;">
                    <div style="width:70px;height:70px;">
                            <div style="width:50px;height:50px;">
                                <div style="width:30px;height:30px;">
                                    <div style="width:10px;height:10px;">
                                    </div>
                                </div>
                            </div>
                    </div>
                </div>
            </div>
        </div>
    </body>
</html>
```

浏览器显示如图 2.17 所示。

图 2.17 例 2-2-1-3 的浏览器显示

例 2-2-1-4 后代选择器应用示例 2。

```
<html>
    <head>
        <title>例 2-14-CSS-多层级-包含</title>
        <style type="text/css">
            div div div div div{border:1px dashed Green;}
              div {margin - left: 10px; margin - top: 10px;
border:1px solid red;}
        </style>
    </head>
    <body>
      <div style="width:130px;height:130px;">
          <div style="width:110px;height:110px;">
                  <div style="width:90px;height:90px;">
                      <div style="width:70px;height:70px;">
                            <div style="width:50px;height:50px;">
                                <form action="">
                                    <div style="width:30px;height:30px;">
                                        <div style="width:10px;height:10px;">
                                        </div>
                                    </div>
                                </form>
                            </div>
                      </div>
                  </div>
          </div>
      </div>
    </body>
```

```
</html>
```

浏览器显示如图 2.18 所示。

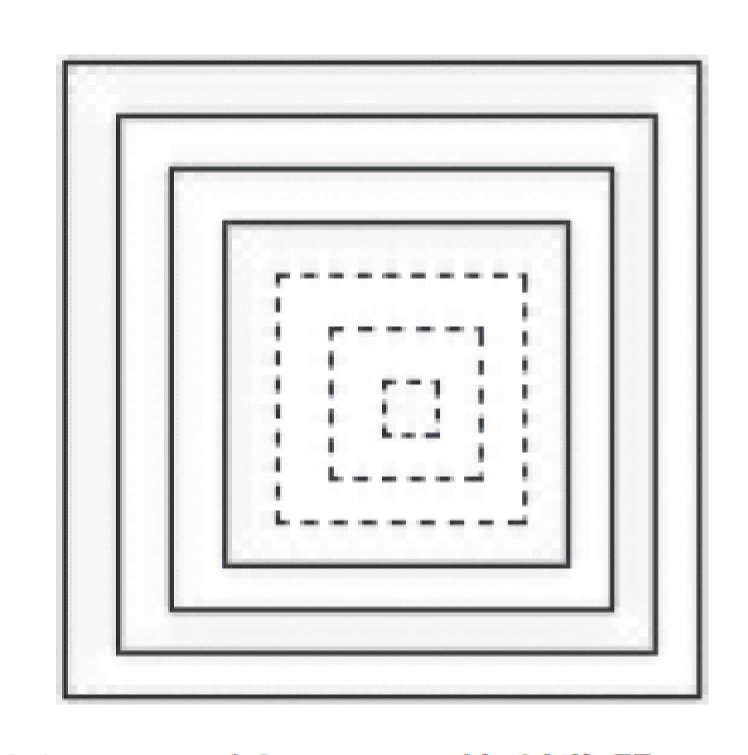


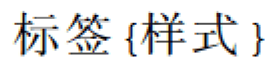
图 2.18　例 2-2-1-4 的浏览器显示

说明：例 2-14 与例 2-13 的区别是在第 5 个和第 6 个 div 之间加了一个 form 标签，但两个例子的浏览器显示相同。因为最内层的 3 个 div 均满足选择条件。故第 5～7 个 div 线型为虚线，颜色为绿色。同样，父代名和后代名可以是标签选择器、类选择器、ID 选择器、属性选择器、属性值选择器、标签属性选择器、标签属性值选择器。

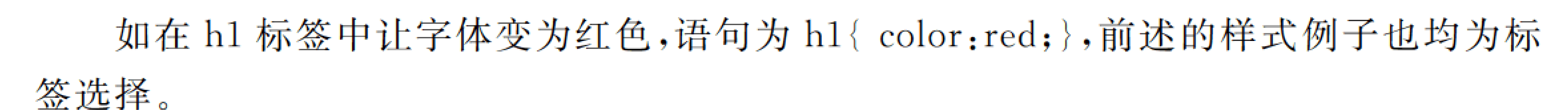
(7) 标签选择器

这是最简单的选择器。

语法：



```
标签{样式}
```



如在 h1 标签中让字体变为红色，语句为 h1{ color:red;}，前述的样式例子也均为标签选择。

(8) ID 选择器

id 选择器可以为标有特定 id 的 HTML 元素指定特定的样式。id 选择器以"#"来定义。

下面两个 id 选择器，第一个定义元素颜色为红色，第二个定义元素颜色为绿色。

```
#summary {color:red;}
#content {color:green;}
```

下面的 HTML 代码中，id 属性为 summary 的 p 元素显示为红色，而 id 属性为 content 的 p 元素显示为绿色。

```
<p id="summary">这个段落是红色。</p>
<p id="content">这个段落是绿色。</p>
```

(9) 类选择器

类选择器以一个点号显示。

```
.class_name{样式规则}
```

body 内的 HTML 标签内使用 class＝"class_name"(class_name 是类选择器名称，可以用任意英文代替，但不能是汉字以及拼音)引用类的样式。

例如：

```
.hp_center {text-align: center}
```

上面例子中所有拥有 hp_center 类的 HTML 元素均为居中。在下面的 BODY 内的 HTML 代码中，h1 和 p 元素都有 hp_center 类。这意味着两者都将遵守.hp_center 选择器中的规则。

```
<h1 class="hp_center">This heading will be center-aligned</h1>
<p class="hp_center">This paragraph will also be center-aligned.</p>
```

注意：类名的第一个字符不能使用数字，因为它无法在 Mozilla 或 Firefox 中起作用。

(10) 属性和值选择器

对带有指定属性的 HTML 元素设置样式。可以为拥有指定属性的 HTML 元素设置样式，而不仅限于 class 和 id 属性。对于 IE7 和 IE8，只有设置!DOCTYPE 才支持属性选择器。在 IE6 及更低的版本中不支持属性选择。

下面的例子为带有 title 属性的所有元素设置样式。

```
[title]{ color:red;}
```

下面的例子为 title="W3School" 的所有元素设置样式。

```
[title=W3School]{ border:5px solid blue; }
```

属性选择器在为不带有 class 或 id 的表单设置样式时特别有用。例如：

```
input[type="text"]
{
    width:150px; display:block; margin-bottom:10px;
    background-color:yellow; font-family: Verdana, Arial;
}
input[type="button"]
{ width:120px; margin-left:35px; display:block; font-family: Verdana, Arial; }
```

2.2.2 创建样式

当读到一个样式表时，浏览器会根据它来格式化 HTML 文档。插入样式表的方法有 3 种：外部样式、内嵌样式和行内样式。

(1) 外部样式表

当样式需要应用于很多页面时，外部样式表将是合理的选择。在使用外部样式表的情况下，可以通过改变一个文件来改变相关网页的外观。每个页面使用<link>标签链接到外部样式表。<link>标签在(文档的)头部，如下所示：

```
<head><link rel="stylesheet" type="text/css" href="mystyle.css" /></head>
```

浏览器会从文件 mystyle.css 中读到样式声明，并根据它来格式文档。外部样式表可以在任何文本编辑器中编辑。文件只能包含样式选择器和样式规则，不能包含任何 html 标签，也不能包含任何 JavaScript 程序。样式表应该以.css 扩展名进行保存。

下面是 mystyle.css 样式表文件的内容：

```
hr {color: sienna;}
p {margin-left: 20px;}
body {background-image: url("images/back40.gif");}
```

不要在属性值与单位之间留有空格。假如使用"margin-left：20 px"而不是"margin-left：20px"，它仅在 IE 6 中有效，但是在 Mozilla/Firefox 或 Netscape 中却无法正常工作。

（2）内嵌样式

当单个文档需要特殊的样式时，应使用内部样式表。使用<style>标签在文档头部定义内部样式表，例如：

```
<head>
    <style type="text/css">
        hr {color: sienna;}
        p {margin-left: 20px;}
        body {background-image: url("images/back40.gif");}
    </style>
</head>
```

（3）行内样式

行内样式将表现和内容混杂在一起，损失样式表的许多优势。它适用于当样式仅需要在一个元素上应用一次时。使用行内样式，需要在相关的标签内使用 style 属性。style 属性可以包含任何 CSS 属性。

例如：

```
<p style="color: sienna; margin-left: 20px">This is a paragraph</p>
```

（4）多重样式

如果某些属性在不同的样式表中被同样的选择器定义，那么属性值将从更具体的样式表中被继承过来。

例如，外部样式表拥有针对 h3 选择器的 3 个属性：

```
h3 { color: red; text-align: left; font-size: 8pt; }
```

而内部样式表拥有针对 h3 选择器的两个属性：

```
h3 { text-align: right; font-size: 20pt; }
```

假如拥有内嵌样式表的页面同时与外部样式表链接，那么 h3 得到的样式如下：

```
color: red; text-align: right; font-size: 20pt;
```

即颜色属性将被继承于外部样式表，而文字排列（text-alignment）和字体尺寸（font-size）会被内部样式表中的规则取代。

同样，拥有内嵌样式表的页面同时具有行内样式，例如：

```
<h3 style='text-align:center;margin-left:20px;'>多重样式</h3>
```

那么 h3 得到的样式是：

```
color: red; text-align: cenetr; font-size: 20pt; margin-left:20px;
```

以上实例说明：当同样的属性被不同的样式表重复定义时，行内样式优先级最高，其次是内嵌样式，外部样式优先级最低。请思考，当同一个样式被不同的选择器重复定义时，优先级情况如何。

2.2.3 创建样式上机实验样例

1. 实验题目

多重样式及优先级验证。

2. 实验代码

(1) CSS 文件。

```
#div1{ border-color:Green; }
.div{ border-style:dotted; }
```

(2) HTML 文件。

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>2-2-2-创建样式</title>
    <link href="2-2-2.css" rel="stylesheet" type="text/css" />
    <style type="text/css">
         .div { border-style:double;border-width:1px;margin-left:10px;margin-
        top:30px;}
        #div1{border-color:Blue;border-width:5px;margin-left:100px;}
    </style>
</head>
<body>
    <div id="div1" class='div' style="border:red;width:70px;height:70px;
    border-width:9px;border-style:groove;"></div>
</body>
</html>
```

3. 浏览器中查看

浏览器中的显示效果如图 2.19 所示。

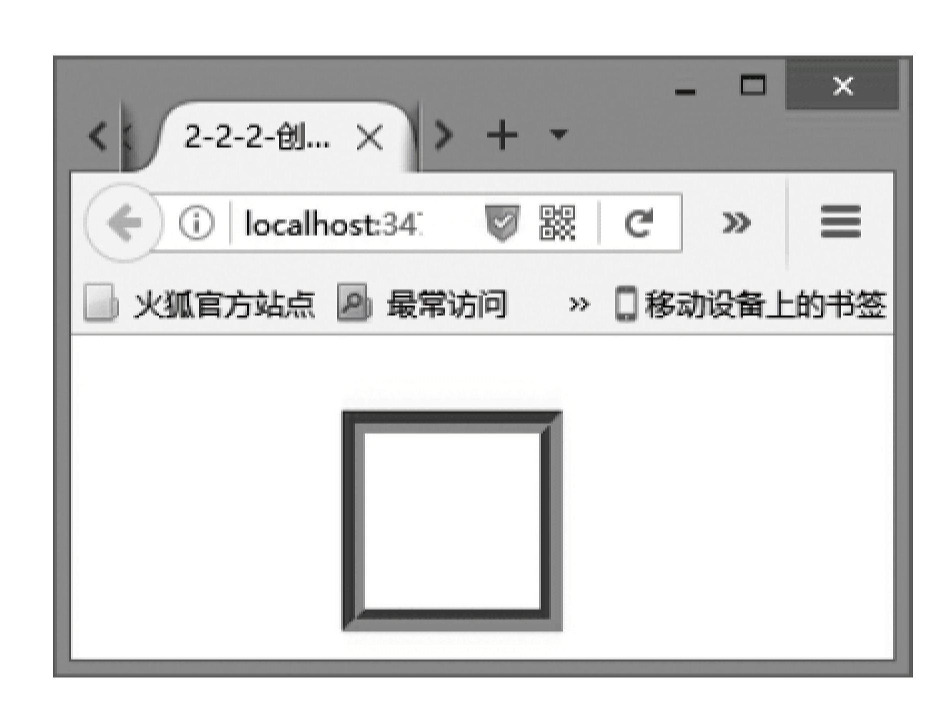


图 2.19 验证 CSS 优先级的浏览器显示

4. 在浏览器的查看器中查看分析运行结果

按 F12 键，浏览器底部出现开发人员工具界面。单击“查看器”，选中 div1，如图 2.20 所示。单击右侧的“规则”，显示界面如图 2.21 所示。



图 2.20　在查看器中查看验证 CSS 优先级文档

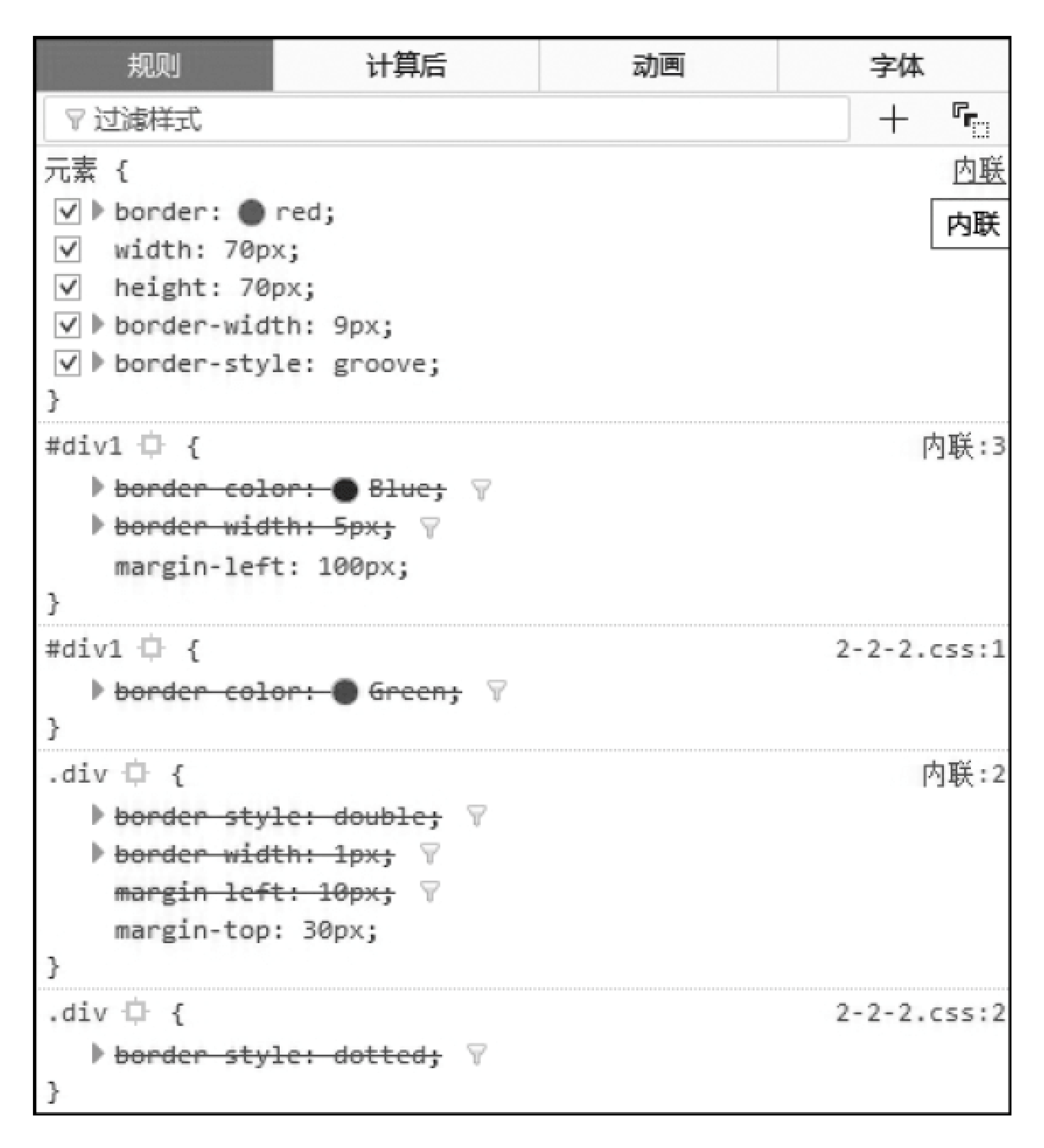


图 2.21　验证 CSS 优先级文档的样式规则

观察图 2.21，得出如下结论：

① 行内(元素)样式优先级最高，在行内的样式全部有效。与行内样式冲突的内嵌 ID 选择器样式和内嵌类选择器样式优先级均低于行内样式。如图 2.5 所示，在内嵌的＃div1 和.div 的 border-width 规则均出现删除线，在外部的＃div1 和内嵌的＃div1 的 border-color 同时出现删除线。同样，与行内样式冲突的外部类和内嵌类选择器样式优先级均低于行内样式。如图 2.5 所示，在外部的.div 和内嵌的.div 的 border-style 规则出现删除线。

② ID 选择器优先级高于类选择器。内嵌的 margin-left 样式规则出现在＃div1 和.div 选择器中，但＃div1 选择器内的样式有效，而.div 选择器内的样式无效，如图 2.21 所示。

(3) 对于某个 HTML 标签,如果有多种样式,且规定的样式没有冲突,则叠加;如果有冲突,则最先考虑行内样式表显示;如果没有行内样式,再考虑内嵌样式显示;如果没有内嵌样式,最后采用外面样式表显示;否则,如果行内样式、内嵌样式和外部样式均没有,按 HTML 的默认样式显示。

2.2.4 样式属性及其描述

1. 背景

CSS 允许应用纯色作为背景,也允许使用背景图像创建复杂效果,背景属性及其描述如表 2.2 所示。

表 2.2 背景属性及其描述

属　性	描　述
background	简写属性,作用是将背景属性设置在一个声明中
background-attachment	背景图像是否固定或者随着页面的其余部分滚动
background-color	设置元素的背景颜色
background-image	把图像设置为背景
background-position	设置背景图像的起始位置
background-repeat	设置背景图像是否及如何重复

所有浏览器都支持 background 属性。background 简写属性在一个声明中设置所有背景属性。

可按 background-color、background-position、background-size、background-repeat、background-origin、background-clip、background-attachment、background-image 的顺序设置。如果不设置其中的某个值,也不会出问题,比如 background:#ff0000 url('smiley.gif); 也是允许的。建议使用这个属性,而不是分别使用单个属性,因为输入的字母也更少。

2. 文本

CSS 文本属性可定义文本外观。通过设置文本属性,可改变文本颜色、字符间距,可对齐文本、装饰文本、对文本进行缩进等,文本属性及其描述如表 2.3 所示。

表 2.3 文本属性及其描述

属　性	描　述
color	设置文本颜色
direction	设置文本方向
line-height	设置行高
letter-spacing	设置字符间距

续表

属　　性	描　　述
text-align	对齐元素中的文本
text-decoration	向文本添加修饰
text-indent	缩进元素中文本的首行
text-shadow	设置文本阴影。CSS2 包含该属性，但是 CSS2.1 没有保留该属性
text-transform	控制元素中的字母
unicode-bidi	设置文本方向
white-space	设置元素中空白的处理方式
word-spacing	设置字间距

3. 字体

CSS 字体属性定义文本的字体系列、大小、加粗、风格（如斜体）和变形（如小型大写字母），相关设置属性及其描述如表 2.4 所示。

表 2.4　字体属性及其描述

属　　性	描　　述
font	简写属性。作用是把所有针对字体的属性设置在一个声明中
font-family	设置字体系列。有两种类型的字体系列名称，包括指定的系列名称和通常字体系列名称。指定的系列名称需要具体字体的名称，如 times、courier、arial。通常字体系列名称如 serif、sans-serif、cursive、fantasy、monospace 等
font-size	设置字体的尺寸
font-size-adjust	当首选字体不可用时，对替换字体进行智能缩放(CSS2.1 已删除该属性)
font-stretch	对字体进行水平拉伸(CSS2.1 已删除该属性)。枚举值：normal、wider、narrower、extra-condensed、condensed、semi-condensed、semi-expanded、expanded、extra-expanded、ultra-expanded
font-style	设置字体风格，枚举值：normal、italic、oblique、inherit
font-variant	以小型大写字体或者正常字体显示文本。枚举值：normal、small-caps 和 inherit
font-weight	设置字体的粗细

所有浏览器都支持 font 属性。任何版本的 Internet Explorer 都不支持属性值 inherit。font 简写属性在一个声明中设置所有字体属性。此属性也有第六个值：line-height，可设置行间距。

这个简写属性用于一次设置元素字体的两个或更多方面。使用 icon 等关键字可以适当地设置元素的字体，使之与用户计算机环境中的某个方面一致。注意，如果没有使用这些关键词，至少要指定字体大小和字体系列。

可按 font-style、font-variant、font-weight、font-size/line-height、font-family 顺序设

置。可以不设置其中的某个值，比如 font:100% verdana; 也是允许的。未设置的属性会使用其默认值。

在一个声明中设置所有字体属性的示例如下：

```
p.ex1 { font:italic arial,sans-serif; }
p.ex2 { font:italic bold 12px/20px arial,sans-serif; }
```

4. 链接

链接能够根据它们所处的状态来设置它们的样式，链接有以下 4 种状态：

a:link：普通的、未被访问的链接。

a:visited：用户已访问的链接。

a:hover：鼠标指针位于链接的上方。

a:active：链接被单击的时刻。

实例如下：

```
a:link {color:#FF0000;}          /* 未被访问的链接 */
a:visited {color:#00FF00;}       /* 已被访问的链接 */
a:hover {color:#FF00FF;}         /* 鼠标指针移动到链接上 */
a:active {color:#0000FF;}        /* 正在被单击的链接 */
```

5. 列表

CSS 列表属性允许放置、改变列表项标志，或者将图像作为列表项标志。

CSS 列表属性及其描述如表 2.5 所示。

表 2.5 CSS 列表属性及其描述

属 性	描 述
list-style	简写属性。用于把所有用于列表的属性设置于一个声明中
list-style-image	将图象设置为列表项标志
list-style-position	设置列表中列表项标志的位置
list-style-type	设置列表项标志的类型

6. 表格

CSS 表格属性可以改善表格的外观，相关属性及其描述如表 2.6 所示。

表 2.6 CSS 表格属性及其描述

属 性	描 述
border-collapse	设置是否把表格边框合并为单一的边框
border-spacing	设置分隔单元格边框的距离

续表

属　　性	描　　述
caption-side	设置表格标题的位置
empty-cells	设置是否显示表格中的空单元格
table-layout	设置显示单元、行和列的算法

7. 轮廓

轮廓绘制元素周围的一条线。该线位于边框边缘的外围，起到突出元素的作用。CSS 的 outline 属性规定元素 4 个外轮廓的样式、颜色和宽度，其描述如表 2.7 所示。outline 属性的示例如下：

```
p  {  outline:#00FF00 dotted thick;  }
```

outline 简写属性在一个声明中设置所有的轮廓属性。可以按顺序设置：outline-color、outline-style 和 outline-width 3 个属性。

表 2.7　轮廓属性及其描述

属　　性	描　　述	CSS
outline	在一个声明中设置所有的轮廓属性	2
outline-color	设置轮廓的颜色	2
outline-style	设置轮廓的样式。枚举值：none、hidden、dotted、dashed、solid、double、groove、ridge、inset、outset、inheri	2
outline-width	设置轮廓的宽度	2

2.2.5　盒子模型

1. 基础知识

规定了元素框处理元素内容、内边距、边框和外边距的方式。在浏览器下，每一个 html 元素都会被解析为一个装有东西的盒子。盒子本身有自己的边框(border)，盒子里的内容到盒子边框的距离称为填充(padding)，盒子边框与其他盒子之间的距离为边界(margin)。在 CSS 模型设计中，元素真实的宽度和高度不仅仅由 width 和 height 来决定，还包括内边距、外边距、边框，如图 2.22 所示。

元素框最内部分是实际内容，直接包围内容的是内边距。内边距呈现了元素的背景。内边距的边缘是边框。边框以外是外边距，外边距默认是透明的，因此不会遮挡其后的任何元素。背景应用于由内容和内边距、边框组成的区域。内边距、边框和外边距都是可选的，默认值是零。但是，许多元素将由用户代理样式表设置外边距和内边距。可以通过将元素的 margin 和 padding 设置为零来覆盖这些浏览器样式。这可以分别进

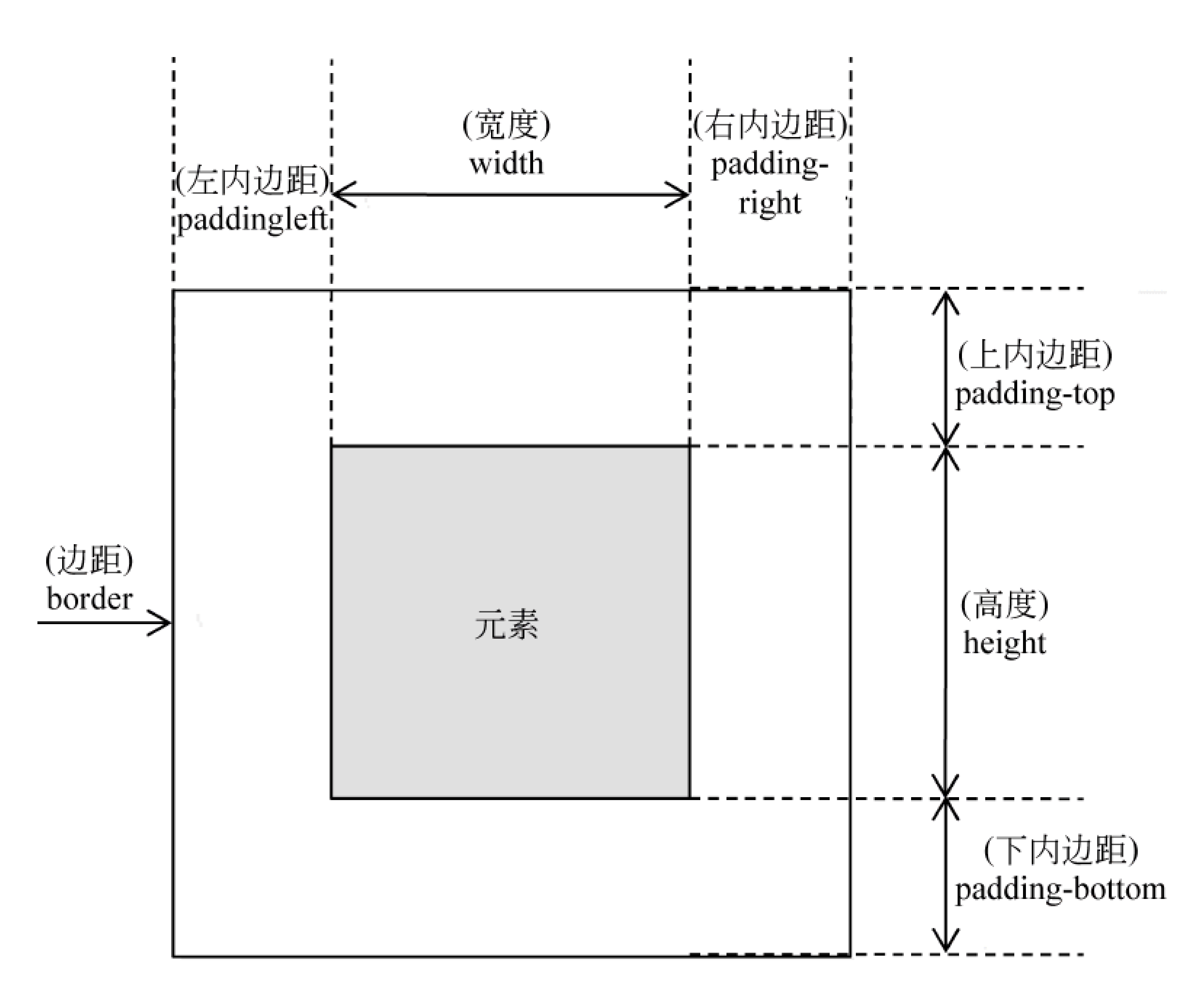


图 2.22　盒子模型

行,也可以使用通用选择器设置所有元素。在 CSS 中,width 和 height 指的是内容区域的宽度和高度。增加内边距、边框和外边距不会影响内容区域的尺寸,但是会增加元素框的总尺寸。内边距、边框和外边距可以应用于一个元素的所有边,也可以应用于单独的边。外边距可以是负值,而且在很多情况下都要使用负值的外边距。

(1) 内边距

元素的内边距在边框和内容区之间。控制该区域最简单的属性是 padding 属性。CSS padding 属性定义元素边框与元素内容之间的空白区域。内边距及其描述如表 2.8 所示。

表 2.8　内边距及其描述

属　　性	描　　述
padding	简写属性。作用是在一个声明中设置元素的内边距属性
padding-bottom	设置元素的下内边距
padding-left	设置元素的左内边距
padding-right	设置元素的右内边距
padding-top	设置元素的上内边距

padding 属性为上、右、下、左 4 个内边距赋值,有多种方式实现,多种方式可以通过复制方式实现简化。简化规则如下:

① 如果缺少左内边距的值,则使用右内边距的值。

② 如果缺少下内边距的值,则使用上内边距的值。

③ 如果缺少右内边距的值,则使用上内边距的值。

例如：

```
h1 { padding: 0.25em 1em 0.5em;}            /* 缺少左内边距,等价于 0.25em 1em 0.5em 1em */
h2 { padding: 0.5em 1em;}          /* 缺少左和下内边距,等价于 0.5em 1em 0.5em 1em */
p { padding: 1px;}                 /* 缺少左、下和右内边距,等价于 1px 1px 1px 1px */
```

(2) 边框

HTML 中使用表格来创建文本周围的边框，但是通过使用 CSS 边框属性可以创建出效果出色的边框，并且可以应用于任何元素。元素外边距内就是元素的边框(border)。元素的边框就是围绕元素内容和内边距的一条或多条线。

每个边框有 3 个方面：宽度、样式及颜色。CSS 规范指出，边框绘制在“元素的背景之上”。因为有些边框是“间断的”(例如点线边框或虚线框)，元素的背景应当出现在边框的可见部分之间。

边框及其描述如表 2.9 所示。

表 2.9　边框及其描述

属　性	描　述
border	简写属性，用于把针对四个边的属性设置在一个声明中
border-style	用于设置元素所有边框的样式，或者单独地为各边设置边框样式。枚举值：none、hidden、dotted、dashed、solid、double、groove、ridge、inset、outset、inheri
border-width	简写属性，用于为元素所有边框设置宽度，或者单独地为各边边框设置宽度
border-color	简写属性，设置元素所有边框中可见部分的颜色，或为 4 个边分别设置颜色
border-bottom	简写属性，用于把下边框的所有属性设置到一个声明中
border-bottom-color	设置元素下边框的颜色
border-bottom-style	设置元素下边框的样式
border-bottom-width	设置元素下边框的宽度
border-left	简写属性，用于把左边框的所有属性设置到一个声明中
border-left-color	设置元素左边框的颜色
border-left-style	设置元素左边框的样式
border-left-width	设置元素左边框的宽度
border-right	简写属性，用于把右边框的所有属性设置到一个声明中
border-right-color	设置元素右边框的颜色
border-right-style	设置元素右边框的样式
border-right-width	设置元素右边框的宽度
border-top	简写属性，用于把上边框的所有属性设置到一个声明中
border-top-color	设置元素上边框的颜色
border-top-style	设置元素上边框的样式
border-top-width	设置元素上边框的宽度

border 属性用于同时设置 4 个边框的宽度、线型和颜色，例如：

```
p  {  border:5px solid red;  }
```

(3) 外边距

围绕在元素边框的空白区域是外边距。设置外边距会在元素外创建额外的“空白”。设置外边距的最简单方法就是使用 margin 属性，这个属性接受任何长度单位（像素、英寸、毫米或 em）、百分数值甚至负值。margin 可以设置为 auto。常见的做法是为外边距设置长度值。margin 的默认值是 0，如果没有为 margin 声明一个值，就不会出现外边距。但在实际中，浏览器对许多元素已经提供了预定的样式，外边距也不例外。例如，在支持 CSS 的浏览器中，外边距会在每个段落元素的上面和下面生成“空行”。因此，如果没有为 p 元素声明外边距，浏览器可能会自己应用一个外边距。当然，作了声明之后，就会覆盖默认样式，外边框属性及其描述如表 2.10 所示。

表 2.10　外边框属性及其描述

属　　性	描　　述
margin	简写属性。在一个声明中设置所有外边距属性
margin-bottom	设置元素下外边距
margin-left	设置元素的左外边距
margin-right	设置元素的右外边距
margin-top	设置元素的上外边距

margin 属性为上、右、下、左 4 个外边距赋值，有多种方式实现，多种方式可以通过复制方式实现简化。简化规则如下：

① 如果缺少左外边距的值，则使用右外边距的值。

② 如果缺少下外边距的值，则使用上外边距的值。

③ 如果缺少右外边距的值，则使用上外边距的值。

例如：

```
h1 {margin: 0.25em 1em 0.5em;}          /* 缺少左外边距,等价于 0.25em 1em 0.5em 1em */
h2 {margin: 0.5em 1em;}          /* 缺少左和下外边距,等价于 0.5em 1em 0.5em 1em */
p {margin: 1px;}          /* 缺少左、下和右外边距,等价于 1px 1px 1px 1px */
```

(4) 外边框合并

外边距合并（叠加）是一个相当简单的概念。但是，在实践中对网页进行布局时，会造成许多混淆。外边距合并是指，当两个垂直外边距相遇时，它们将形成一个外边距。合并后的外边距高度等于两个发生合并的外边距的高度中的较大者。当一个元素出现在另一个元素上面时，第一个元素的下外边距与第二个元素的上外边距会发生合并，如图 2.23 所示。

当一个元素包含在另一个元素中时（假设没有内边距或边框把外边距分隔开），它们的上和/或下外边距也会发生合并，如图 2.24 所示。

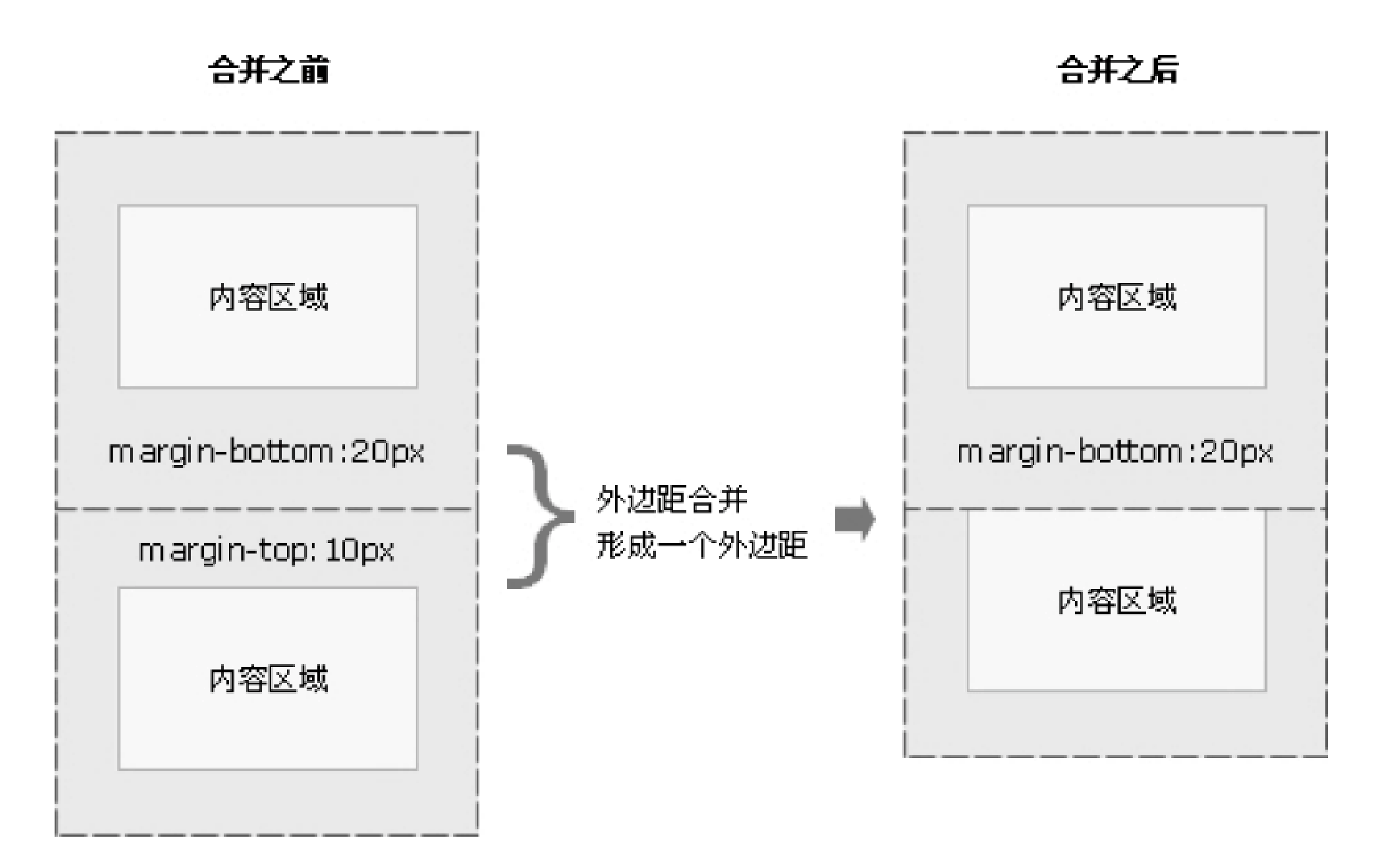


图 2.23　一个元素的下外边距与另一个元素的上外边距合并

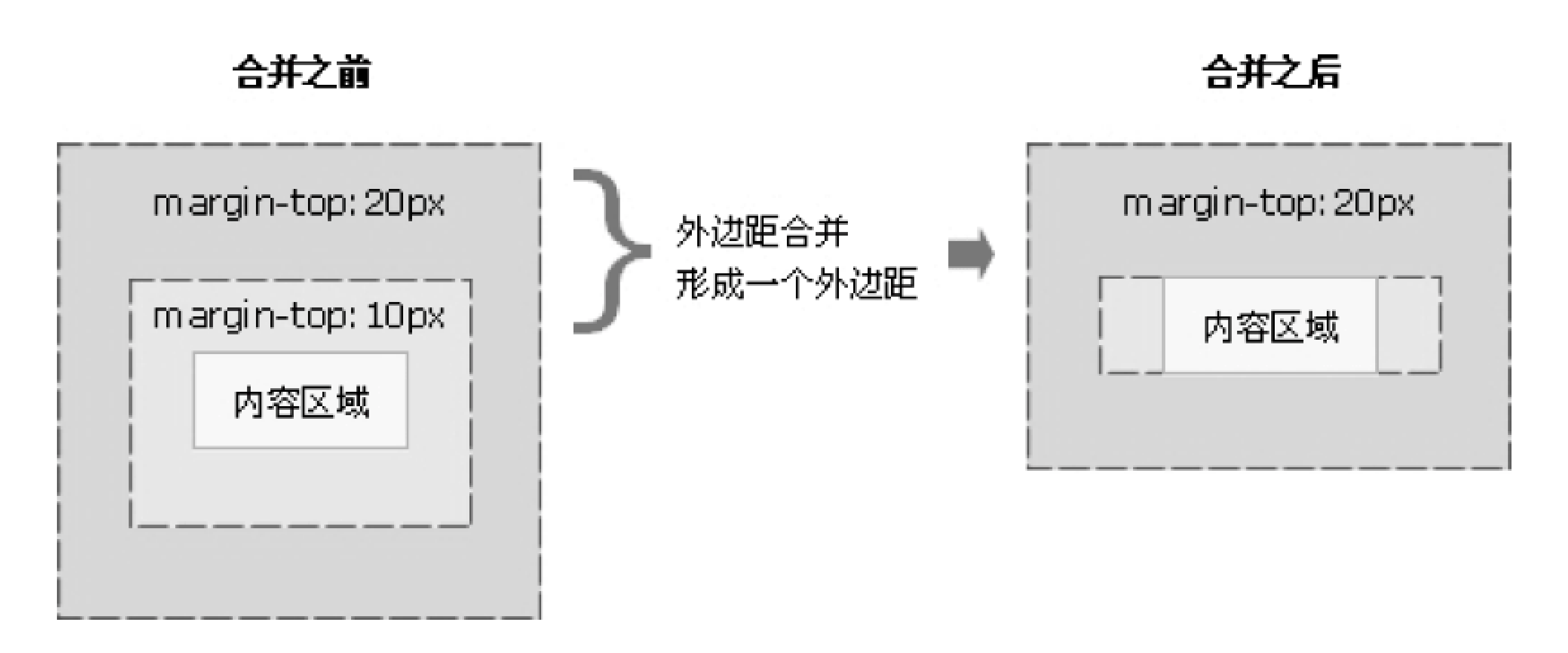


图 2.24　被包含元素与包含元素的上外边距合并

假设有一个空元素，它有外边距，但是没有边框或填充。此时上外边距与下外边距就碰到了一起，它们会发生合并，如图 2.25 所示。

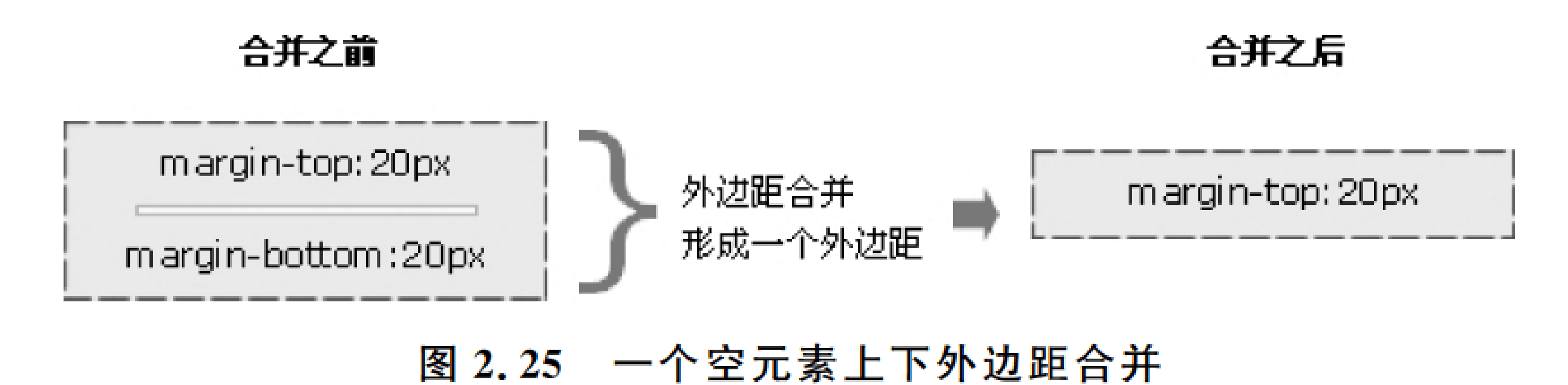


图 2.25　一个空元素上下外边距合并

如果这个外边距遇到另一个元素的外边距，还会发生合并，如图 2.26 所示。

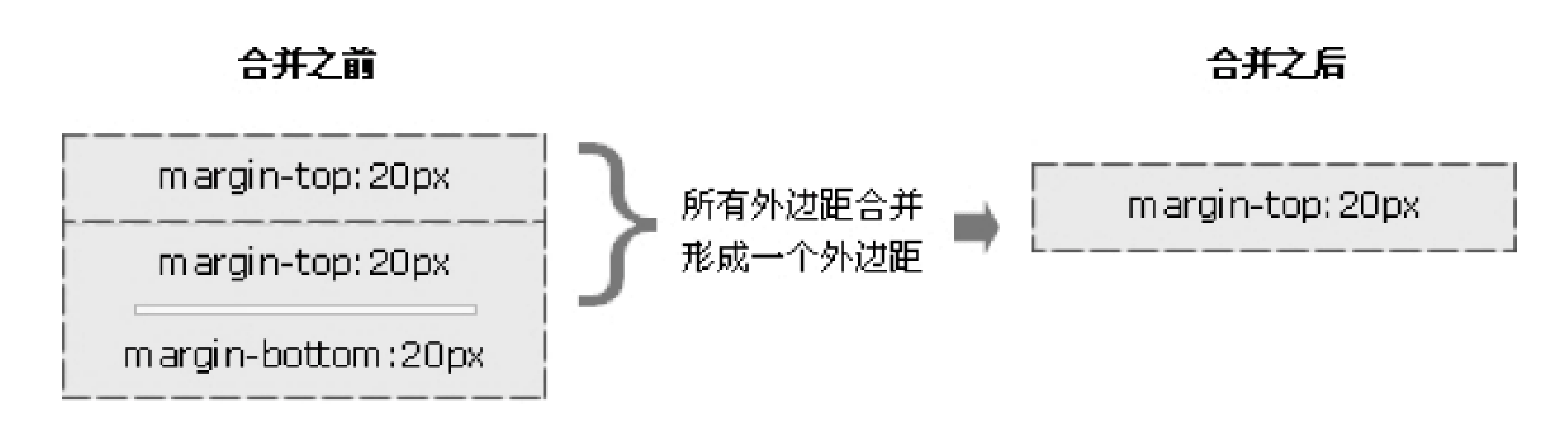


图 2.26　一个空元素与父元素外边距合并

这就是一系列的段落元素占用空间非常小的原因，因为它们的所有外边距都合并到一起，形成了一个小的外边距。

外边距合并具有实际意义。以由几个段落组成的典型文本页面为例。第一个段落上面的空间等于段落的上外边距。如果没有外边距合并，后续所有段落之间的外边距都将是相邻上外边距和下外边距的和。这意味着段落之间的空间是页面顶部的两倍。如果发生外边距合并，段落之间的上外边距和下外边距就合并在一起，这样各处距离就一致了。

段落文档上下间距均匀的示例如图 2.27 所示。

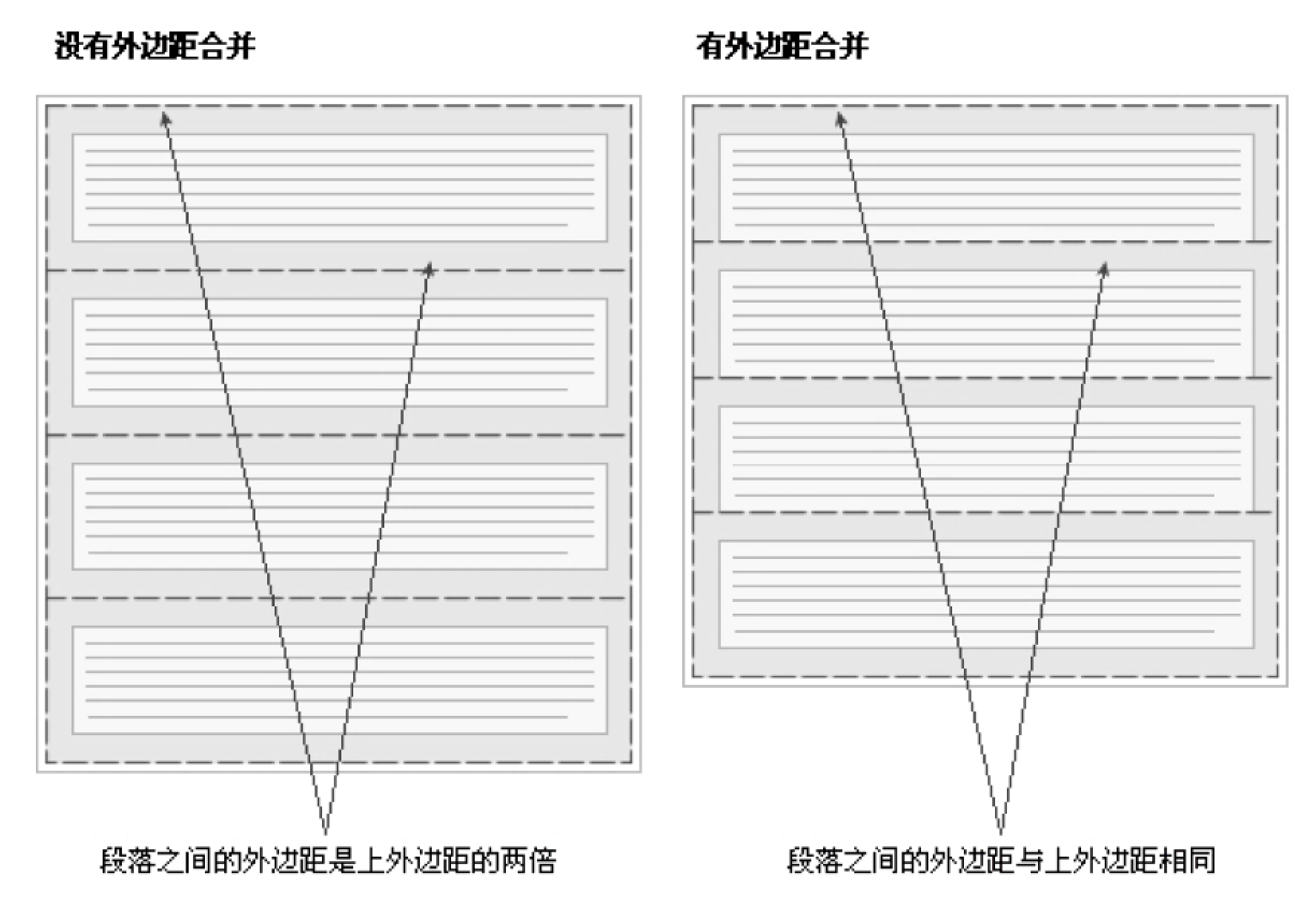


图 2.27　段落文档上下间距均匀的示例

只有普通文档流中块框的垂直外边距才会发生外边距合并。行内框、浮动框或绝对定位之间的外边距不会合并。

2. DIV 标签

<DIV>定义文档中的分区和节，能把文档分割为独立的、不同的部分，可用做严格的组织工具。如果用 ID 或者 Class 来标记<DIV>，那么该标签的作用会变得更加有效。<DIV>是一个块级元素。这意味着它的内容自动地开始一个新行。实际上，换行是<DIV>固有的唯一格式表现。可以通过<DIV>的 Class 或 ID 应用额外的样式。不必为每一个<DIV>都加上类或 ID，虽然这样做也有一定的好处。可以对同一个<DIV>元素应用 Class 或 ID 属性，但是更常见的情况是只应用其中一种。这两者的主要差异是 Class 用于元素组（类似的元素，或者可以理解为某一类元素），而 ID 用于标识单独的唯一的元素。div 全局属性及其描述如表 2.11 所示。

表 2.11　div 全局属性及其描述

属　　性	HTML5 新增	描　　述
accesskey		规定激活元素的快捷键
class		规定元素的一个或多个类名(引用样式表中的类)
contenteditable	是	规定元素内容是否可编辑
contextmenu	是	规定元素的上下文菜单。上下文菜单在用户单击元素时显示
data- *	是	用于存储页面或应用程序的私有定制数据
dir		规定元素中内容的文本方向
draggable	是	规定元素是否可拖动
dropzone	是	规定在拖动被拖动数据时是否进行复制、移动或链接
hidden	是	规定元素仍未或不再相关
id		规定元素的唯一 id
lang		规定元素内容的语言
spellcheck	是	规定是否对元素进行拼写和语法检查
style		规定元素的行内 CSS 样式
tabindex		规定元素的 Tab 键次序
title		规定有关元素的额外信息
translate	是	规定是否应该翻译元素内容

2.2.6　盒子模型上机实验样例

1. 实验目标

验证盒子的外形尺寸。

2. 程序功能

建立一个 HTML 文档,用嵌套的 div 实现盒子的模拟。通过观察 2 个 div 的外形尺寸分析盒子模型外形宽和高与 width、height、border、padding 的关系。

3. 实验步骤

(1) 建立源程序:2-2-4-盒子模型.htm

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>2-2-4 盒子模型</title>
<style type="text/css">
```

```
.box1{ border:1px solid #666666;
        width:500px;
        height:50px;
        background:#CCCC00;
        margin:30px
      }
.box2{ border:1px dashed #FF0000;
        width:300px;
        padding:10px 20px;
        color:#ffffff;
        background-color:#000066;
        margin:20px 10px 20px -30px;
}
</style>
</head>
   <body>
      <div class="box1">盒子 1<div class="box2">盒子 2</div></div>
   </body>
</html>
```

(2) 用 Firefox 浏览器浏览 2-2-4-盒子模型.htm

浏览器结果如图 2.28 所示。



图 2.28 在浏览器中浏览 2-2-4-盒子模型.htm

(3) 按 F12 键用开发人员工具的查看器观察

在图 2.28 中按 F12 键,浏览器的底部出现开发人员工具,激活"查看器"活页夹,如图 2.29 所示。

① 观察 div.box1。

单击查看器的 div.box1 元素,浏览器中给出其外形尺寸为 502×52,如图 2.30 所示。

分析:外形宽=border-left+padding-left+width+padding-right+border-right,选中开发人员工具的规则,看到 width=500px;border:1px solid #666666,没有 padding 意味着 padding-left 和 padding-right 均为 0px,border 为 1px 意味着 border-left 和



图 2.29 在浏览器的查看器中查看 2-2-4-盒子模型.htm

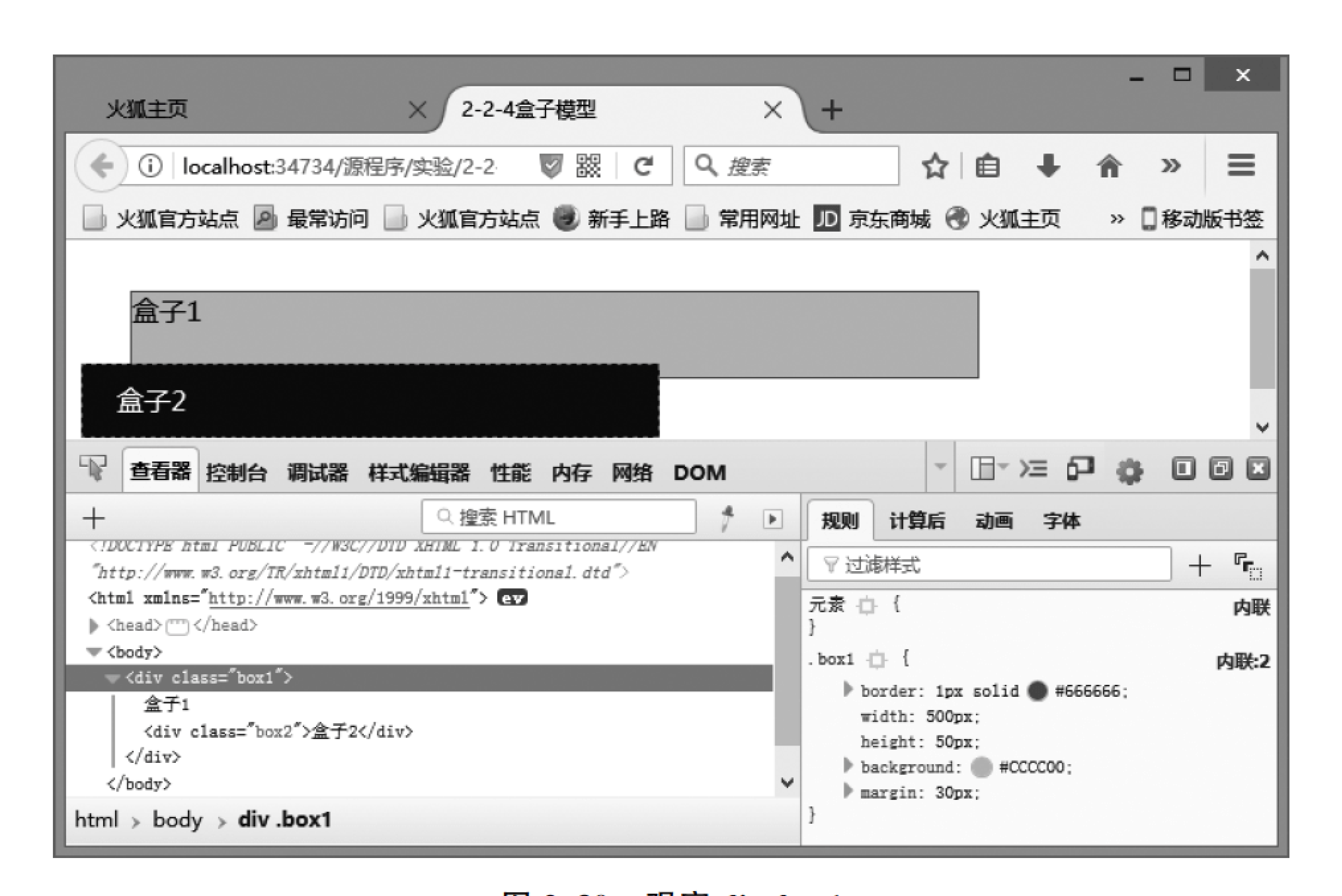


图 2.30 观察 div. box1

border-right 均为 1px。因此，502＝1＋0＋500＋0＋1；同理，外形高＝border-top＋padding-top＋height＋padding-bottom＋border-bottom，52 的由来类似于 502 的分析。

② 观察 div. box2。

展开 div1，单击查看器的 div. box2 元素，则浏览器中给出其外形尺寸为 342×44，如图 2.31 所示。图 2.31 与图 2.30 在外形尺寸上的区别是图 2.31 中的规则中有 padding，

但没有 height。padding：10px 20px 意味着 padding-top 和 padding-bottom 为 10px，而 padding-left 和 padding-right 为 20px；没有 height 意味着内容高度用系统默认的 22px，分别代入外形高与外形宽的公式，会得到 44 和 342。



图 2.31 观察 div.box2

2.2.7 CSS 定位与浮动

1. 定义

CSS 定位属性允许对元素进行定位。CSS 为定位和浮动提供了一些属性，利用这些属性可以建立列式布局，将布局的一部分与另一部分重叠，还可以完成多年来通常需要使用多个表格才能完成的任务。

定位定义元素框相对于其正常位置应该出现的位置，或者相对于父元素、另一个元素甚至浏览器窗口本身的位置。另一方面，CSS1 中就提出了浮动，它以 Netscape 在 Web 发展初期增加的一个功能为基础。浮动不完全是定位，不过，它当然也不是正常流布局。

2. 块级元素与行内元素

div、h1 或 p 元素常常被称为块级元素。这意味着这些元素显示为一块内容，即“块框”。与之相反，span 和 strong 等元素称为“行内元素”，因为其内容显示在行中，称为“行内框”。

可使用 display 属性改变生成的框的类型。将 display 属性设置为 block，可以让行内元素(比如 <a> 元素)表现得像块级元素一样。还可以把 display 设置为 none，使生成的元素根本没有框。这样该框及其所有内容就不再显示，不占用文档中的空间。

但在一种情况下，即使没有进行显式定义，也会创建块级元素。这种情况发生在把

一些文本添加到一个块级元素(比如 div)的开头。即使没有把这些文本定义为段落,它也会被当做段落对待。在这种情况下,这个框称为无名块框,因为它不与专门定义的元素相关联。

块级元素的文本行也会发生类似的情况。假设有一个包含 3 行文本的段落,每行文本形成一个无名框。无法直接对无名块或行框应用样式,因为没有可以应用样式的地方(注意,行框和行内框是两个概念)。但这有助于理解在屏幕上看到的所有东西都形成某种框。

3. 定位机制

CSS 有 3 种基本的定位机制:普通流、浮动和绝对定位。除非专门指定,否则所有框都在普通流中定位。也就是说,普通流中元素的位置由元素在(X)HTML 中的位置决定。块级框从上到下一个接一个地排列,框之间的垂直距离是由框的垂直外边距计算出来的。

行内框在一行中水平布置,可使用水平内边距、边框和外边距调整它们的间距。但是,垂直内边距、边框和外边距不影响行内框的高度。由一行形成的水平框称为行框(Line Box),行框的高度总是足以容纳它包含的所有行内框。不过,设置行高可以增加这个框的高度。

4. position 属性

使用 position 属性可以选择 4 种不同类型的定位,这会影响元素框生成的方式。position 属性值的含义如下。

(1) static。

元素框正常生成。块级元素生成一个矩形框,作为文档流的一部分,行内元素则会创建一个或多个行框,置于其父元素中。

(2) relative。

元素框偏移某个距离。元素仍保持其未定位前的形状,它原本所占的空间仍保留。如果对一个元素进行相对定位,它将出现在它所在的位置上。然后可以通过设置垂直或水平位置,让这个元素"相对于"它的起点进行移动。

如果将 top 设置为 20px,那么框将在原位置顶部下面 20 像素之处。如果 left 设置为 30 像素,那么元素左边会创建 30 像素的空间,也就是将元素向右移动,如图 2.32 所示。

```
#box_relative {  position: relative;  left: 30px;  top: 20px; }
```

使用相对定位时,无论是否移动,元素仍然占据原来的空间。因此,移动元素会导致它覆盖其他框。

(3) absolute。

元素框从文档流完全删除,并相对于其包含块定位。包含块可能是文档中的另一个

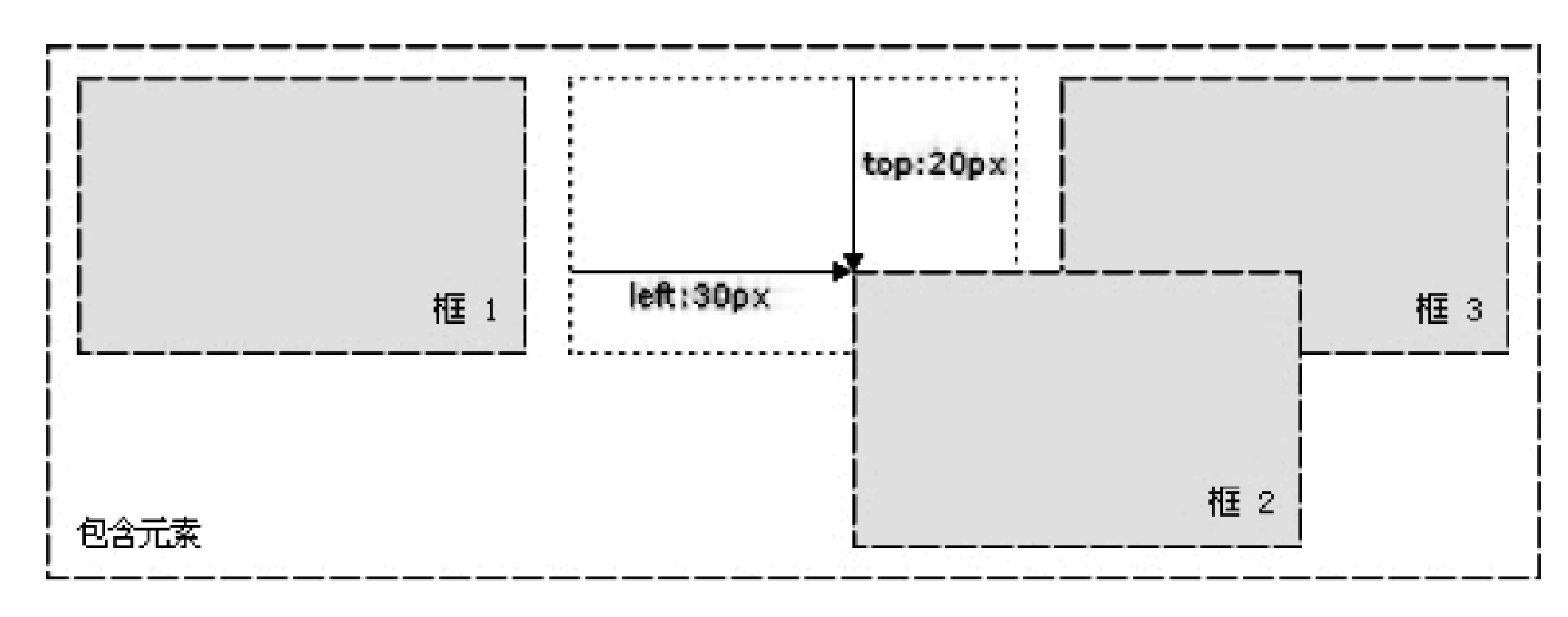


图 2.32 相对定位示例

元素或者是初始包含块。元素原先在正常文档流中所占的空间会关闭，就好像元素原来不存在一样。元素定位后生成一个块级框，而不管它原来在正常流中生成何种类型的框。绝对定位使元素的位置与文档流无关，因此不占据空间。这一点与相对定位不同，相对定位实际上被看做普通流定位模型的一部分，因为元素的位置相对于它在普通流中的位置。普通流中其他元素的布局就像绝对定位的元素不存在一样，如图 2.33 所示。

```
#box_relative { position: absolute; left: 30px; top: 20px; }
```

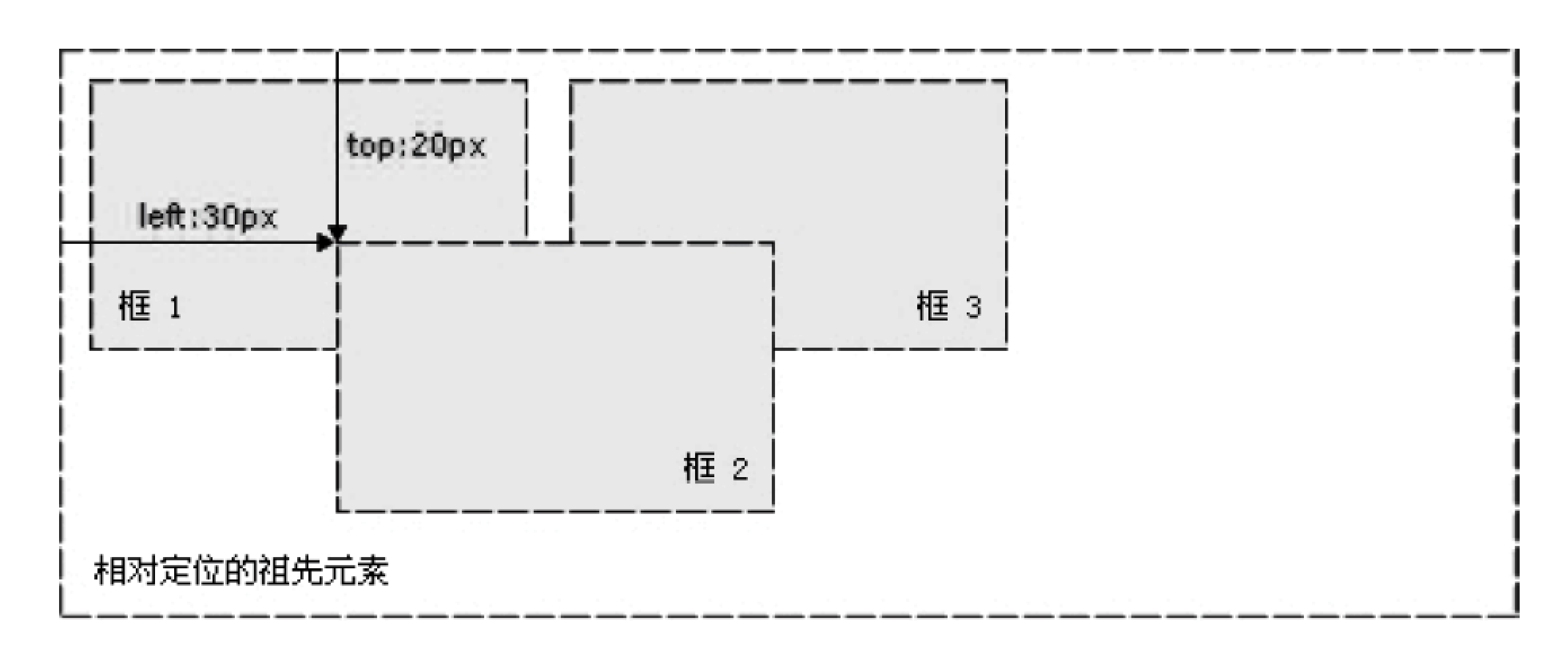


图 2.33 绝对定位示例

绝对定位元素的位置相对于最近的已定位祖先元素，如果元素没有已定位的祖先元素，那么它的位置相对于最初的包含块。对于定位的主要问题，是要记住每种定位的意义。相对定位是“相对于”元素在文档中的初始位置，而绝对定位是“相对于”最近的已定位祖先元素，如果不存在已定位的祖先元素，那么“相对于”最初的包含块。根据用户代理的不同，最初的包含块可能是画布或 HTML 元素。因为绝对定位的框与文档流无关，所以它们可以覆盖页面上的其他元素。可以通过设置 z-index 属性来控制这些框的堆放次序。

（4）fixed。

元素框的表现类似于将 position 设置为 absolute，不过其包含块是视窗本身。

定位属性及其描述如表 2.12 所示。

表 2.12 定位属性及其描述

属 性	描 述
position	把元素放置到一个静态的、相对的、绝对的或固定的位置中
top	定义了一个定位元素的上外边距边界与其包含块上边界之间的偏移
right	定义了定位元素右外边距边界与其包含块右边界之间的偏移
bottom	定义了定位元素下外边距边界与其包含块下边界之间的偏移
left	定义了定位元素左外边距边界与其包含块左边界之间的偏移
overflow	设置当元素的内容溢出其区域时发生的事情。枚举值：visible、hidden、scroll、auto 和 inherit
clip	设置元素的形状。元素被剪入这个形状之中，然后显示出来
vertical-align	设置元素的垂直对齐方式
z-index	设置元素的堆叠顺序

5. 浮动

浮动的框可以向左或向右移动，直到它的外边缘碰到包含框或另一个浮动框的边框为止。由于浮动框不在文档的普通流中，所以文档的普通流中的块框表现得就像浮动框不存在一样。当把框 1 向右浮动时，它脱离文档流并且向右移动，直到它的右边缘碰到包含框的右边缘，如图 2.34 所示。

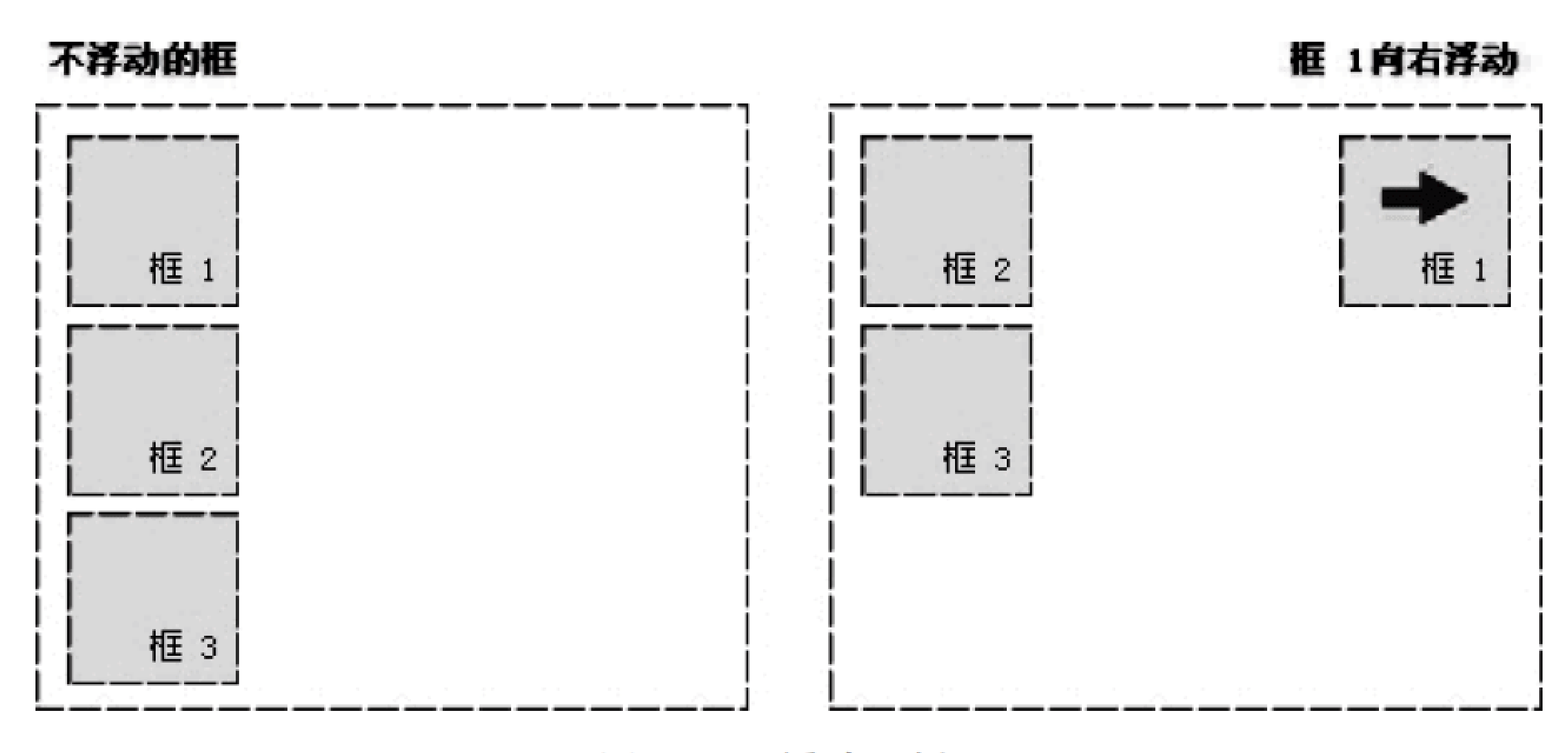


图 2.34 浮动示例 1

当框 1 向左浮动时，它脱离文档流并且向左移动，直到它的左边缘碰到包含框的左边缘。因为它不再处于文档流中，所以不占据空间，实际上覆盖住了框 2，使框 2 从视图中消失。如果把所有 3 个框都向左移动，那么框 1 向左浮动直到碰到包含框，另外 2 个框向左浮动直到碰到前一个浮动框，如图 2.35 所示。

如图 2.36 所示，如果包含框太窄，无法容纳水平排列的 3 个浮动元素，那么其他浮动块向下移动，直到有足够的空间。如果浮动元素的高度不同，那么当它们向下移动时，可能被其他浮动元素“卡住”。

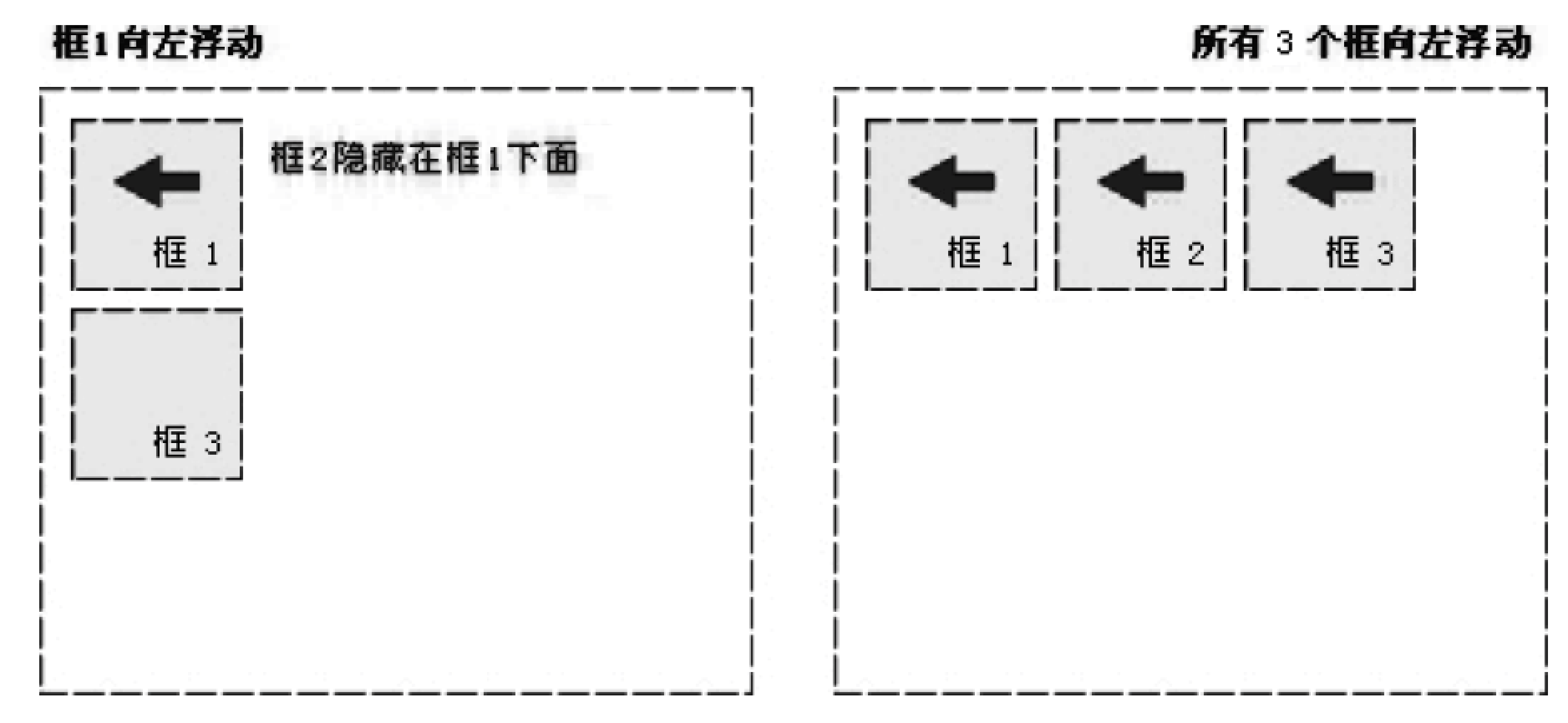


图 2.35 浮动示例 2

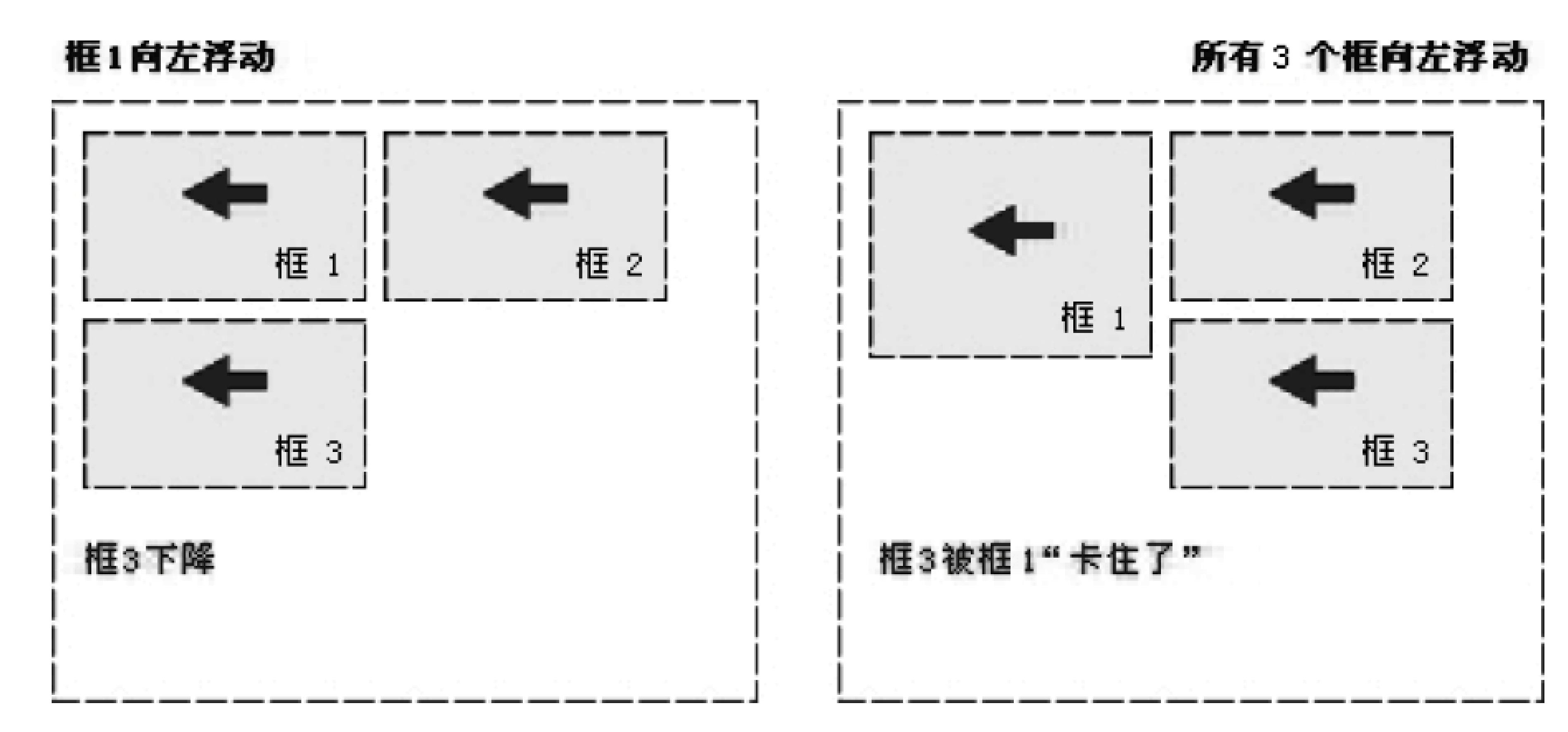


图 2.36 浮动示例 3

通过 float 属性实现元素的浮动。所有主流浏览器都支持 float 属性。float 属性定义元素的浮动方向。在 CSS 中,任何元素都可以浮动。浮动元素会生成一个块级框,而不论它本身是何种元素。如果浮动非替换元素,则要指定一个明确的宽度;否则,它们会尽可能地窄。假如一行之上只有极少的空间可供浮动元素,那么这个元素会跳至下一行,这个过程会持续到某一行拥有足够的空间为止。

float 枚举值及其描述如表 2.13 所示。

表 2.13 float 枚举值及其描述

值	描 述
left	元素向左浮动
right	元素向右浮动
none	默认值。元素不浮动,并会显示在其在文本中出现的位置
inherit	规定应该从父元素继承 float 属性的值

6. 水平导航条的创建方法

有两种创建水平导航栏的方法。使用行内或浮动列表项。两种方法都不错,如果希

望链接拥有相同的尺寸，须使用浮动方法。构建水平导航栏的方法之一是将<li>元素规定为行内元素，语句如下：

```
li{display:inline;} a{display:block;width:60px;}
```

为了让所有链接拥有相等的宽度，浮动<li>元素并规定<a>元素的宽度：

```
li{float:left;}
a{display:block;width:60px;}
```

其中，a{display:block;width:60px;} 把链接显示为块元素可使整个链接区域可单击(不仅仅是文本)，同时也允许规定宽度。

2.2.8 定位与浮动上机实验样例

1. 实验目标

掌握运用浮动制作水平菜单。

2. 程序功能

建立一个 HTML 文档，运用 div、ul、li、a 标签，结合 CSS，引用 float 属性，完成水平菜单设计任务。

3. 实验步骤

(1) 源程序-水平横向菜单.htm

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>水平横向菜单</title>
    <style type="text/css">
        #layout{background-color:#FFFFFF; height:300px; width:800px; border:
        2px solid #A9C9E2}
        #www{list-style-type:square;list-style-image:url(img/leaf.png);}
        #file{list-style-type:square;list-style-image:url(img/profile.png);}
        #email{list-style-type:square;list-style-image:url(img/search.png);}
        #telnet{list-style-type:square;list-style-image:url(img/tags.png);}
        #other{list-style-type:square;list-style-image:url(img/truck.png);}
        a:hover{color:#f00;}
        #layout ul li {float: left;width: 140px;text-align: center;}
    </style>
</head>
<body>
    <div id='layout'>
        <ul>
            <li id='www'><a href="#" target="_blank">WWW服务</a></li>
            <li id='file'><a href="#" target="_blank">文件传输服务</a></li>
```

```
            <li id='email'><a href="#" target="_blank">电子邮件服务</a></li>
            <li id='telnet'><a href="#" target="_blank">远程登录服务</a></li>
            <li id='other'><a href="#" target="_blank">其他服务</a></li>
        </ul>
    </div>
</body>
</html>
```

(2) 在浏览器中浏览

在浏览器中浏览的结果如图 2.19 所示。

(3) 在浏览器的查看器中调试运行结果

在图 2.37 所示的浏览器中按 F12 键，浏览器底部弹出开发人员工具界面，选中“查看器”选项卡，选择第一个＃layout ul li 元素，鼠标指向＃layout ul li 规则，如图 2.38 所示。在图 2.38 中取消＃layout ul li 规则的 float 属性后，界面如图 2.39 所示。重新选中 float 属性，返回到图 2.38 的状态。



图 2.37　水平横向菜单示例



图 2.38　“查看器”中查看＃layout ul li 样式规则

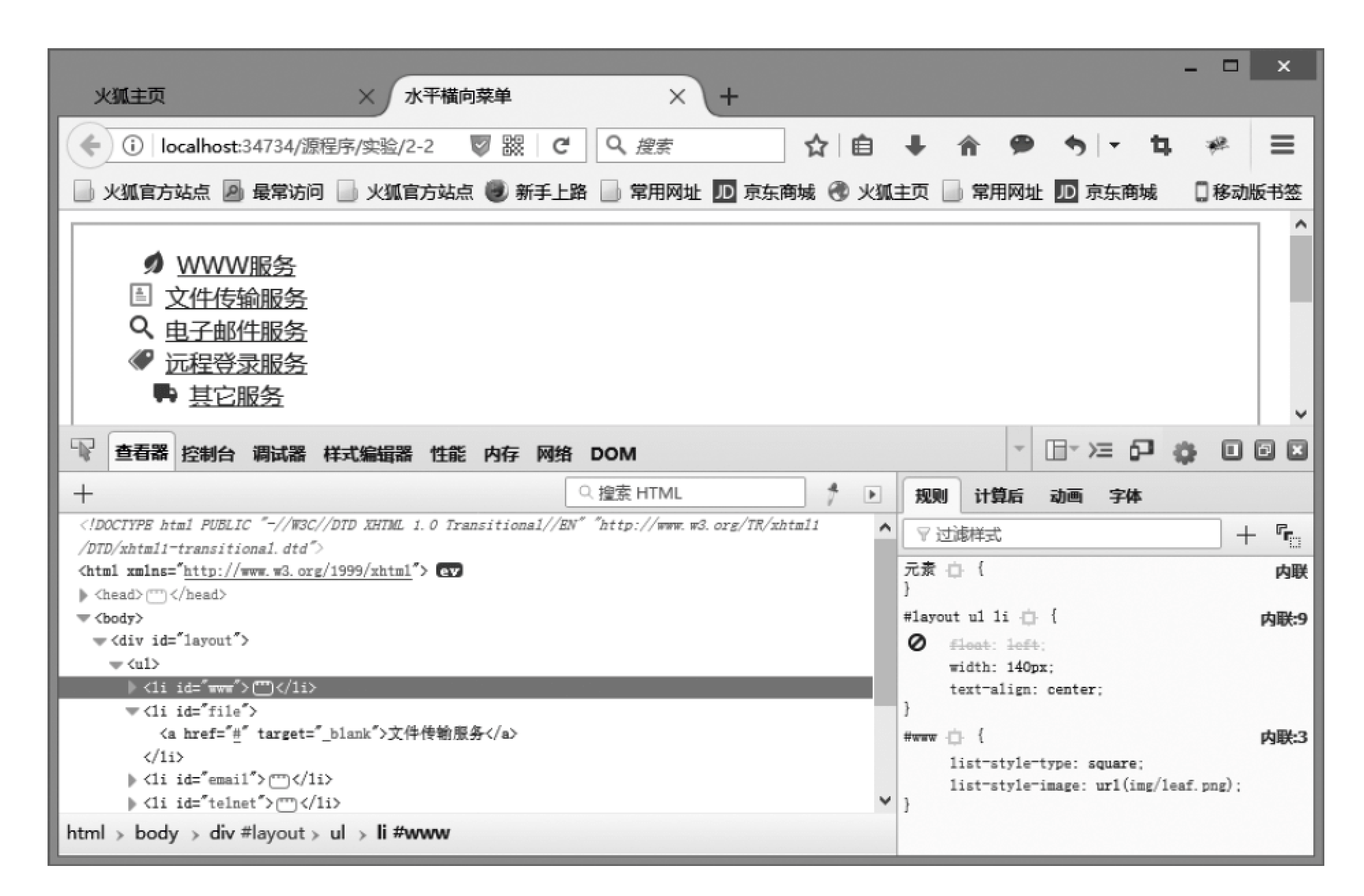


图 2.39 取消“查看器”中#layout ul li 样式规则的 float 属性后

2.3 实验任务

1. 实验题目

基于 DIV 的画图首页布局。

2. 程序功能

综合利用所学的标签和 CSS 设计一个画图网页,含横向水平主菜单、纵向树形面板和客户区。横向水平主菜单含有长方形、圆、菱形、正方形、椭圆 5 个菜单项。纵向树形面板含有线型、颜色、动静 3 个选项,线型含有实线、虚线 2 个子项,颜色有红、绿、蓝 3 个子项,动静含有动和静 2 个子选项。

3. 实验类型

综合设计。

4. 实验要求

(1) 至少 3 个 div,分别为主菜单区、树形面板区和客户区,使用绝对定位。
(2) 比例布局合理。
(3) 含有外部样式、内嵌样式和行内样式。
(4) 横向水平菜单均匀等间距等长分布。
(5) 各个菜单项均含有图标。

5. 实验环境

(1) 计算机：PC、内存 8GB、主频 1.8GHz 及以上、硬盘 500GB 及以上。
(2) 操作系统：Windows XP、Windows 7、Windows 8、Windows 10。
(3) 开发环境：Visual Studio 2010 或 Adobe Dreamweaver。
(4) 浏览器：IE8 及以上、Chrome、Firefox、Safari、Edge、QQ 浏览器等。

6. 实验原理

(1) 盒子模型。
(2) 图文列表。
(3) 定位与浮动。

7. 源代码

(1) 辅以必要的注释。
(2) 错落有致、结构清晰、易读性强。

8. 遇到的问题及解决办法

第 3 章 chapter 3

HTML 布局

■ **知识目标**

- 掌握布局的方式、属性和工具
- 理解常用的布局结构
- 掌握流式布局
- 掌握 bootstrap 进行布局所用的基本控件

■ **能力目标**

- 能够根据实际需求选择恰当的布局方式、结构
- 能够根据实际选取合适的开发工具
- 能够灵活运用所选的开发工具设计切题的、新颖的、丰富的页面

■ **素质目标**

- 运用 bootstrap 快速开发响应式页面

■ **教学重点**

- 运用 Bootstrap 进行布局

■ **教学难点**

- 设计精美精致精准的网页布局

■ **建议学时**

- 理论：3 学时
- 实验：3 学时

3.1 概　　述

1. 布局方式

布局有自然布局、响应布局和定位布局 3 种方式。自然布局没有任何修饰，自动靠左。响应布局为流动布局，采用浮动方式。定位布局有相对定位和绝对定位。

相对定位和绝对定位都是相对于父 div 标签的。相对定位以这个元素本来应该在的

位置为参照点。绝对定位以父 div 标签的原点(左上角)为参照点。由于外层是 position:relative,所以如果里层是 absolute,则会以外层的左上角为位移参考对齐。当然,外层只写 position:relative,写上 left、top 这两个值,则表示以这个元素本来应该在的位置为布局参照原点进行 left、top 对齐。

如果只有一个 position:absolute,外层没有 position:relative,此时的处理原则是如果某父级元素中有 relative,则以某父级元素为参考原点,如果没有 position:relative,则以 body 为参考原点。如果 position:absolute 外层没有 relative,这两个布局是没有区别的。

最后,如果外层是 position:absolute,内层是 position:relative,absolute 会参考 body 为布局原点,relative 会参考它本来应该在的位置为布局原点,其实是参考外层左上角为布局原点。

2. 布局结构

HTML5 有 3 种典型布局结构,分别是 div-ul-li、table-tr-td 和 frameset-frame。div-ul-li 常用于分类导航或菜单等场合。table-tr-td 常用于图文布局或显示数据的场合。frameset-frame 用于一个浏览器窗口显示多个网页。

3. 布局属性

无论采用何种布局方式和布局结构,都涉及表 3.1 所示的 CSS 常用布局属性功能。

表 3.1 CSS 布局常用属性功能

属　性	功　能
position	绝对定位样式实现 DIV 定位布局,值设为 absolute 或 relative
width	宽度,设置对象宽度
height	高度,设置对象高度
line-height	行高,设置文本行高
left	设置 div 对象靠左距离
right	设置 div 对象靠左距离
top	设置 div 对象靠左距离
bottom	设置 div 对象靠左距离
background	背景,设置背景图片和颜色
color	设置字体颜色
font-size	设置字体大小
font-weight	设置字体加粗
font-family	设置字体

4. 布局工具

布局工具有 div、table、frameset、HTML5 新元素和 Bootstrap。前三者为 HTML 标

签，而 Bootstrap 是一个用于快速开发 Web 应用程序和网站的前端框架。

3.2 div-ul-li 布局实例

一个典型的 div+CSS 布局实例在 firefox 浏览器的显示效果如图 3.1 所示。



图 3.1　div+CSS 布局实例的浏览器显示

1. 实例分析

(1) 选择 ul+li 列表

此 CSS+DIV 布局实例是典型的列表型布局，可在每个列表项 li 的上面放一个图片，下面放一个标题。

(2) 设置图片盒子模型

图片使用 img 标签，宽度 160px、高度 90px、display 设为级块。

(3) 设置标题盒子模型

标题文字使用 span 盒子包裹，同时设置 span 背景色、宽度 100%、高度 30px、display 设为块级。

(4) 设置列表项 li 盒子模型

list-style-type 为 none，float 为 left，padding 为 4px、8px，width 为 160px。设置好列表项 li 宽度并且浮动靠左，可让列表项 li 并列显示，一排显示不完自动换行，继续显示列表项 li 的图文内容。每个 li 上下和左右各有相同间距，使用内边距 padding 设置列表项的间距。

(5) 设置无序列表 ul 盒子模型

取消无序列表 ul 的内边距 padding 和外边距 margin 的默认值，设置其 padding 为

0px，margin 为 0px，width 为 528，overflow 为 hidden。

2. 准备图片素材

将图片放入 img 文件夹。

3. 详细 HTML 文档

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        div.layout{border:2px solid #CCCCCC; padding:3px; width:528px;
        height:256px;}
        ul.imglist{ padding:0px;margin:0px; width:528px; overflow:hidden}
        ul.imglist li{ list-style-type:none; float:left; padding:4px 8px;
        width:160px}
        ul.imglist li img{ display:block; width:160px; height:90px}
        ul.imglist li span{display:block; width:100%; height:30px; line-
        height:30px; background:#F6F6F6; text-align:center;}
    </style>
</head>
<body>
    <div class="layout">
        <ul class="imglist">
            <li>
                <a href="#" target="_blank">
                  <img src="img/W_2015-11-2-17-20-59.jpg" alt=""/>
                  <span>瓷砖展示 1</span>
                </a>
            </li>
            <li>
                <a href="#" target="_blank">
                    <img src="img/W_2015-11-2-17-22-54.jpg"  alt=""/>
                  <span>瓷砖展示 2</span>
                </a>
            </li>
            <li>
                <a href="#" target="_blank">
                    <img src="img/W_2015-11-2-17-27-4.jpg"  alt=""/>
                  <span>瓷砖展示 3</span>
                </a>
            </li>
            <li>
                <a href="#" target="_blank">
                    <img src="img/W_2015-11-2-17-33-58.jpg"  alt=""/>
                  <span>瓷砖展示 4</span>
```

```
                </a>
            </li>
            <li>
                <a href="#" target="_blank">
                    <img src="img/W_2015-11-2-17-34-32.jpg"  alt=""/>
                  <span>瓷砖展示 5</span>
                </a>
            </li>
            <li>
                <a href="#" target="_blank">
                    <img src="img/W_2015-11-2-17-40-22.jpg"   alt=""/>
                  <span>瓷砖展示 6</span>
                </a>
            </li>
        </ul>
    </div>
</body>
</html>
```

4. 层叠样式表 CSS 解释

（1）无序列表图片类

```
ul.imglist{ padding:0px;margin:0px auto; width:528px; overflow:hidden}
```

设置 padding:0px，使用 margin:0 auto，让 ul 结构布局居中，使用 overflow:hidden。这样可以解决子级使用 float 属性产生的不能撑开问题。同时，使用固定宽度 width:528px(528=(160+8×2)×3)，使得一排只排 3 个 li。

（2）无序列表图片类列表项

```
ul.imglist li{ list-style-type:none; float:left; padding:4px 8px; width:160px}
```

使用 list-style-type:none 取消标题文字的项目符号；float:left，让 li 靠左开始并列；padding:4px 8px 设置 li 与 li 之间的间距；width:160px 设置固定宽度。

（3）无序列表图片类列表项中的图片

```
ul.imglist li img{ display:block; width:160px; height:90px}
```

display:block 设置图片独占一行；width:160px; height:90px 设置图片固定高度宽度。

（4）无序列表图片类列表项中行内元素

```
ul.imglist li span{display:block; width:100%; height:30px; line-height:30px; background:
#F6F6F6}
```

display:block 让 span 设置宽度与高度生效时，文字独占一行效果；width:100%；height:30px；设置宽度和高度，宽度 100%会继承父级 li 宽度（等于 160px 宽度），设置 100%宽度的好处是随父级宽度而自动计算出 100%宽度与父级宽度保持一致；line-height:30px；设置文字在 30px 中垂直居中；background:#F6F6F6 设置 span 背景颜色。

5. HTML 关键点

使用 ul+li 组合列表标签布局,每个 li 盒子放图片+文字标题,li 盒子直接使用 a 超链接标签,将图片和文字内容包裹于其中。使用 img 标签引入图片,使用 span 标签显示标题。span 与 img 形成两个盒子,用 CSS 分别设置独占一行的样式(display:block),避免文字和图片排成一排而非上下结构。

6. 浏览器调试观察

观察 1:div. layout 的尺寸。

如图 3.2 所示,div. layout 的尺寸为 538×266,根据盒子模型:538=528+(3+2)×2,266=256+(3+2)×2。



图 3.2 开发人员工具的查看器观察 div. layout 尺寸

观察 2:ul. imglist 的尺寸。

如图 3.3 所示,ul. imglist 的尺寸为 528×256,根据盒子模型:528=528+(0+0)×2,256=256+(0+0)×2。

观察 3:取消 ul. imglist 的 padding 设置。

如图 3.4 所示,取消 ul. imglist 的 padding 设置后,ul. imglist 向右偏移,原因是取消后浏览器采用默认的 ul. imglist 的 padding 设置。



图 3.3 开发人员工具的查看器观察 ul. imglist 尺寸



图 3.4 取消 ul. imglist 的 padding 设置后浏览器的显示

3.3 table-tr-td 布局实例

一个典型的 table+CSS 布局实例在 firefox 浏览器的显示效果如图 3.5 所示。

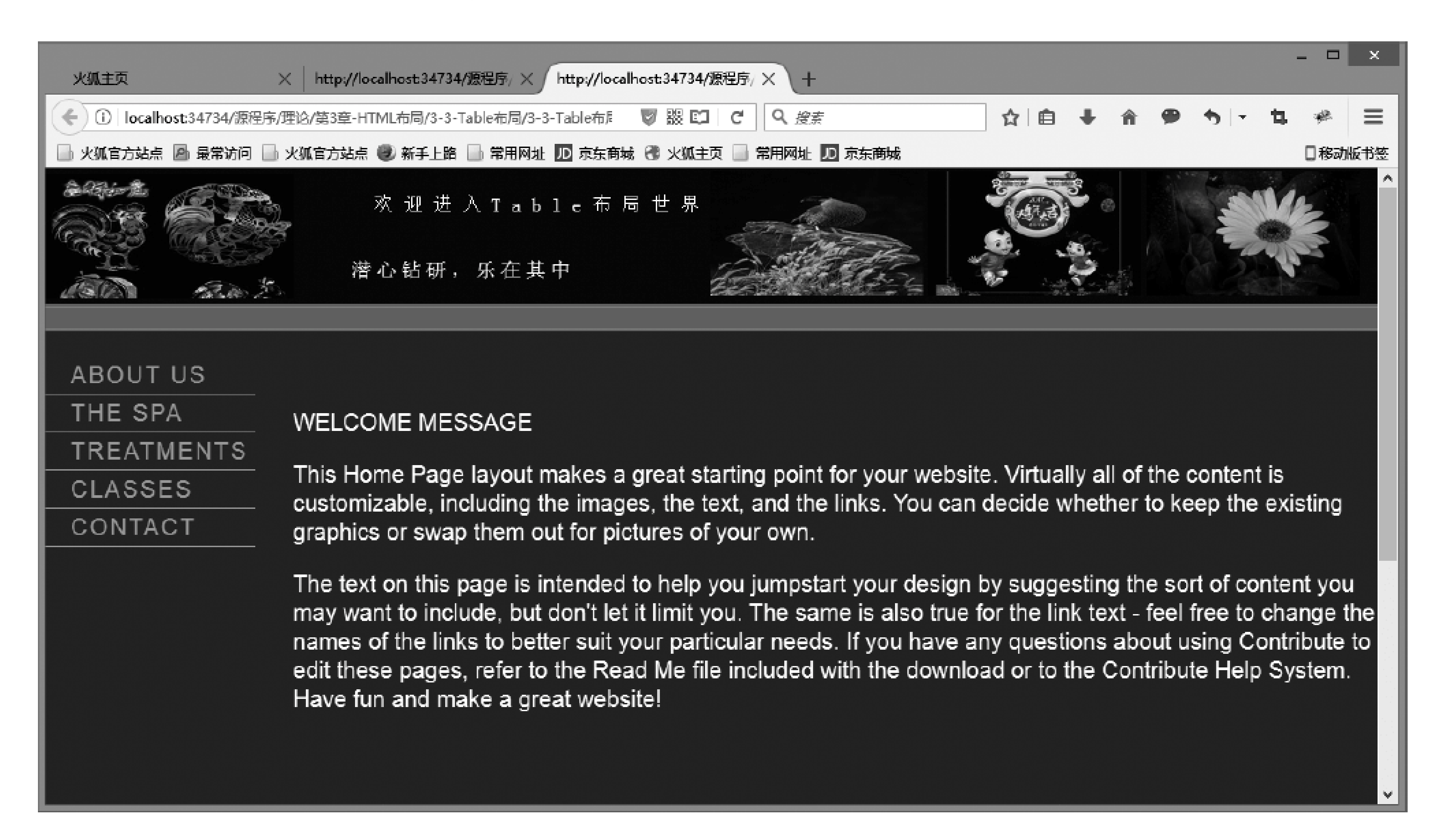


图 3.5 Table+CSS 布局实例的浏览器显示

1. 实例分析

本实例的分区均基于表进行，无论一级分区还是二级分区。一级分区指直接隶属于 body 元素的 table，该表被称为主表。二级分区指隶属于主表内一个 td 里的表。

(1) 一级分区

图 3.5 所示的网页从上向下分顶部标题和 logo 区、横分隔、操作展示区。操作展示区从左向右分纵向菜单区、纵分隔、文本展示区。因此，主表分块设计如图 3.6 所示。主表各区行列分配情况如表 3.2 所示。

<table>
<tr><td colspan="2" rowspan="2">logo</td><td>标题</td><td rowspan="2">图片</td></tr>
<tr><td>提示</td></tr>
<tr><td colspan="2"></td><td>横分隔</td><td></td></tr>
<tr><td>纵向菜单区</td><td>纵分隔</td><td colspan="2">文本展示区</td></tr>
</table>

图 3.6 主 Table 分块设计

表 3.2　主表各区行列分配情况

区　　名	行数/跨行	列数/跨列
logo	2/是	2/是
标题	1/否	2/是
提示	1/是	2/是
图片	2/是	1/否
横分隔	5/否	5/是
纵向菜单区	1/否	1/否
纵分隔	1/否	1/否
文本展示区	1/否	3/是

(2) 二级分区

纵向分隔区和文本展示区运用二级分区进行细化。纵向分隔区用表占 5 行 1 列。每行显示一个菜单项。每个 td 的 border-bottom 显示为 1px，起到下划线的效果。文本展示区用表占 2 行 1 列。第 1 行为标题，第 2 行含 2 个段落元素。

2. 准备图片素材

将图片放入 img 文件夹。

3. 详细 HTML 文档

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        body { margin:0px; }
        #logo  {
         font:17px Arial, Helvetica, sans-serif;
         color: white;
         letter-spacing:.5em;
         line-height:30px;
        }
        #tagline  {
         font:18px Arial, Helvetica, sans-serif;
         color: white;
         letter-spacing:.2em;
         line-height:14px;
        }
        #navigation td { border-bottom: 1px solid #FF9900 }
        #navigation a {
```

```
            font:20px Arial, Helvetica, sans-serif;
            color: #FF9900;
            line-height:16px;
            letter-spacing:.1em;
            text-decoration: none;
            display:block;
            padding:8px 6px 8px 22px;
            }
        #navigation a:hover {
            color:black;
            font-weight:bold;
            background-color: #FFFACD;
            }
        .pageName,.bodyText{color: white;}
    </style>
</head>
<body style="background-color:#7e0202">
    <table cellspacing="0" cellpadding="0">
        <tr bgcolor="#220103">
            <td width="215" rowspan="2" colspan="2">
                <img src="img/1.jpg" alt="Header image" width="215" height="108"
                border="0" />
              </td>
                <td width="355" colspan="2" height="54" nowrap="nowrap" id="logo"
                valign="bottom">
                                            <pre>    欢迎进入 Table 布局世界</pre>
              </td>
              <td width="176" rowspan="2">
                  <img src="img/2.jpg" alt="header image 2" width="185" height="108"
                  border="0"/>
                  <img src="img/3.jpg" alt="header image 2" width="185" height="108"
                  border="0"/>
                  <img src="img/4.jpg" alt="header image 2" width="185" height="108"
                  border="0"/>
              </td>
          </tr>
          <tr bgcolor="#220103">
              <td height="54" colspan="2" nowrap="nowrap" id="tagline"
              valign="top">
                  <pre>    潜心钻研,乐在其中</pre>
              </td>
          </tr>
            <tr bgcolor="#FF080E"><td colspan="5" style="height: 2px"></td>
              </tr>
```

```
<tr bgcolor="#FF9900"><td colspan="5" style="height: 1px"></td>
    </tr>
<tr bgcolor="#FF080E"><td colspan="5" style="height: 18px"></td>      </
tr>
<tr bgcolor="#FF9900"><td colspan="5" style="height: 1px"></td>
    </tr>
<tr bgcolor="#FF080E"><td colspan="5" style="height: 2px"></td>
    </tr>
<tr>
    <td width="170" height="700" valign="top" id="navborder"><br />
        <table border="0" cellspacing="0" cellpadding="0" width=
        "170" id="navigation">
          <tr><td width="165"><a href="" class="navText">ABOUT US</a><
          /td></tr>
          <tr><td width="165"><a href="" class="navText">THE SPA</a></
          td></tr>
          <tr><td width="165"><a href="" class="navText">TREATMENTS</a
          ></td>
          </tr>
          <tr><td width="165"><a href="" class="navText">CLASSES</a></
          td></tr>
          <tr><td width="165"><a href="" class="navText">CONTACT</a></
          td></tr>
        </table>
    </td>
    <td width="45"></td>
    <td width="100%" colspan="3" valign="top">
        <br/><br/><br/>
            <table border="0" cellspacing="0" cellpadding="0" width="100%"
            style="background-color:#7e0202;font:20px Arial, Helvetica,
            sans-serif;">
             <tr>
               <td class="pageName">WELCOME MESSAGE</td>
             </tr>
             <tr>
               <td class="bodyText">
                  <p>This Home Page layout makes a great starting point for
                     your website. Virtually all of the content is
                     customizable, including the images, the text, and the
                     links. You can decide whether to keep the existing
                     graphics or swap them out for pictures of your own.
                  </p>
                 <p>The text on this page is intended to help you jumpstart
                    your design by suggesting the sort of content you may
```

```
                                    want to include, but don't let it limit you. The same is
                                    also true for the link text - feel free to change the
                                    names of the links to better suit your particular needs.
                                    If you have any questions about using Contribute to edit
                                    these pages, refer to the Read Me file included with the
                                    download or to the Contribute Help System. Have fun and
                                    make a great website!
                                </p>
                              </td>
                            </tr>
                        </table>
                        <br/><br/>
                    </td>
                  </tr>
        </table>
    </body>
    </html>
```

4. 知识点

(1) <td> 标签的 nowrap 属性

规定单元格中的内容不换行,属性行为与 table、td 的 width 属性无关。

(2) em

em 指字体高,任意浏览器的默认字体高都是 16px,1em＝16px。那么,12px＝0.75em,10px＝0.625em。为了简化 font-size 的换算,需要在 CSS 中的 body 选择器中声明 Font-size＝62.5%,这就使 em 值变为 16px * 62.5%＝10px,这样 12px＝1.2em,10px＝1em,也就是说只需要将原来的 px 数值除以 10,然后换上 em 作为单位就行了。em 有如下特点:

① em 值并不是固定的。

② em 继承父级元素字体大小。

5. 技术要点

(1) 表格总宽度不固定,具体宽度由单元格 100% 的列设置补充,整个宽度充满屏幕。

(2) 页边距在样式表中由标签 body 设置。

(3) 字体在样式表中由标签 #logo、#tagline 设置。

(4) 布局表格属性在网页中设置<table border＝"0" cellspacing＝"0" cellpadding＝"0">。

(5) 在单元格中插入图片时,单元格的宽度应该由图片宽度决定,两个图片的高度应该一致。

(6) 背景在<body>标签中设置<body style＝"background-color:#7e0202>,也

可使用<body bgcolor="#990000" background="mm_bg_red. gif">设置。

3.4 frameset布局实例

1. 基础知识

使用框架可以在同一个浏览器窗口中显示多个页面。每份 HTML 文档称为一个框架,并且每个框架都独立于其他框架。使用框架的缺点是开发人员必须同时跟踪更多的 HTML 文档,而且很难打印整张页面。

框架结构标签(<frameset>)定义如何将窗口分割为框架。每个 frameset 定义了一系列行或列,rows/columns 的值规定了每行或每列占据屏幕的面积。框架标签(Frame)签定义了放置在每个框架中的 HTML 文档。在 frameset 里使用 iframe 来实现常规的一些分栏布局,可以借助一个页面承载多个页面的方式来重用页面代码。一个页面中的 iframe 之间可以互相实现关联,而且不需要过多依靠 JavaScript 就可以实现类似"局部"刷新的机制。要让不同 iframe 之间的 dom 元素产生互动和关联,需要 JavaScript 程序复杂程度较高,而且有些根本实现不了(比如拖曳、用户自定制页面布局等)。一个页面承载多个页面的 HTTP 请求,实现类似局部刷新的效果,不是后台运行请求机制,而是前台浏览器刷新机制,会有延迟的响应,而且无法对响应进行控制(比如 loading 或者一些响应状态的监听)。Frameset 适用于具有固定布局的系统,但 HTML 5 已不支持这个标签。

下面的例子中设置了一个两列的框架集。第一列被设置为占据浏览器窗口的 25%。第二列被设置为占据浏览器窗口的 75%。HTML 文档"frame_a. htm"被置于第一个列中,而 HTML 文档"frame_b. htm"被置于第二个列中。

```
<frameset cols="25%,75%">
    <frame src="frame_a.htm">
    <frame src="frame_b.htm">
</frameset>
```

假如一个框架有可见边框,可以拖动边框来改变它的大小。为了避免这种情况发生,可以在 <frame> 标签中加入 noresize="noresize"。

iframe 用于在网页内显示网页。添加 iframe 的语法是<iframe src="URL"></iframe>,URL 指向隔离页面的位置。Iframe 可设置高度和宽度,由 height 和 width 属性规定属性值,属性值的默认单位是像素,但也可以用百分比来设定(比如 "80%")。例如:

```
<iframe src="demo_iframe.htm" width="200" height="200"></iframe>
```

Iframe 能够显示或隐藏边框,由 frameborder 属性控制,设置属性值为 "0" 就可以移除边框,设置为非 0 显示边框。例如:

```
<iframe src="demo_iframe.htm" frameborder="0"></iframe>
```

iframe 可用做链接的目标(target),由 target 和 name 属性联合控制。例如:

```
<iframe src="demo_iframe.htm" name="iframe_a"></iframe>
<p><a href="http://www.w3school.com.cn" target="iframe_a">W3School.com.cn
  </a></p>
```

2. 布局实例

一个典型的 frameset 布局实例在 firefox 浏览器的显示效果如图 3.7 所示。

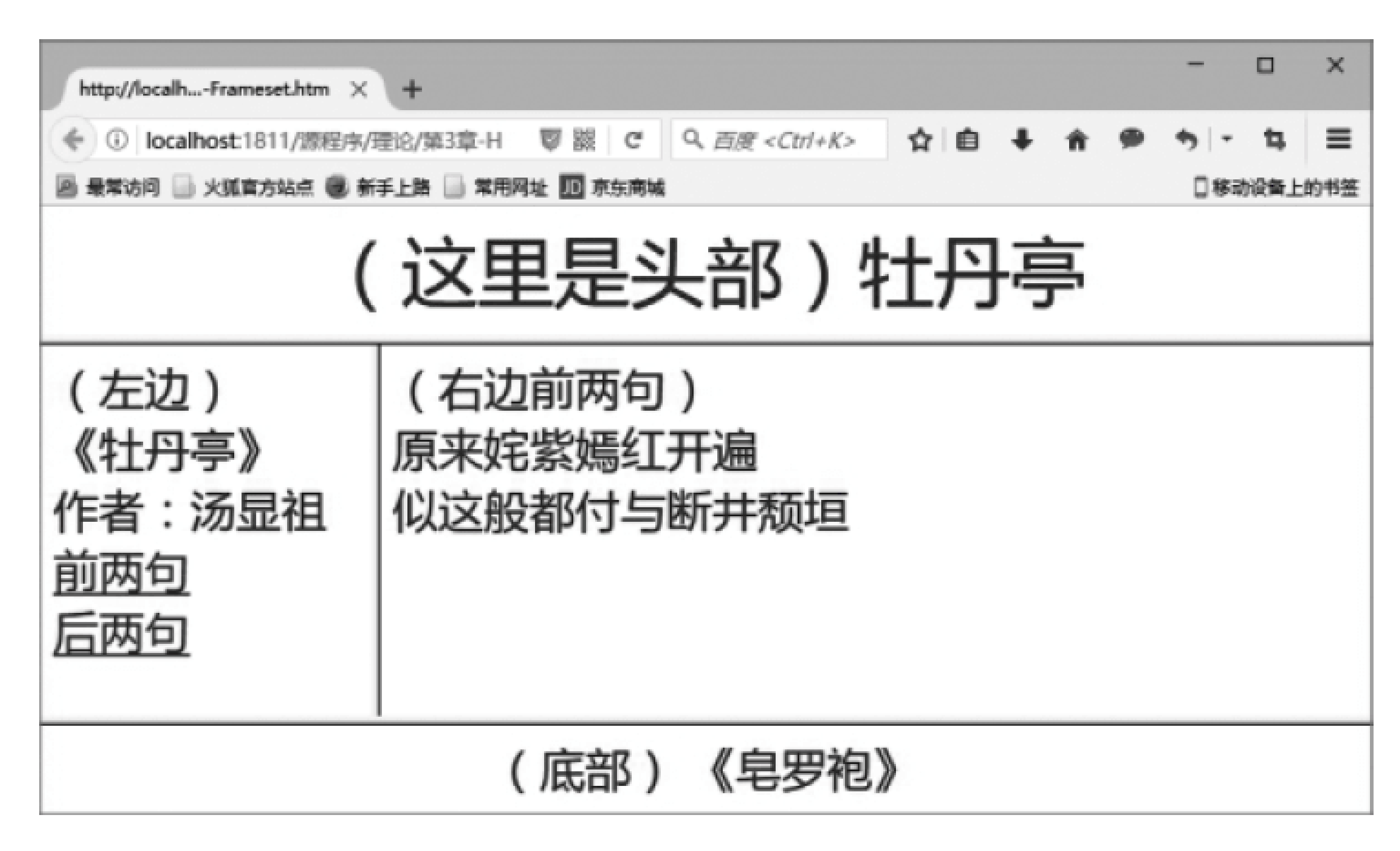


图 3.7 frameset 布局实例

1) 实例分析

本实例进行两级分区,一级分区为主 frameset,把浏览器水平分隔成三行,分别占浏览器的 15%、75%和 10%。二级分区是把第二行垂直分成左右两部分,分别占 25%和 75%。二级分区的左边至少含有如图 3.7 所示的内容,其中前两句和后两句为超链接,链接 HTML 文档目标为本分区的右边部分。

2) 详细 HTML 文档

有 6 个 html 文档文件,文件名分别为 3-4-Frameset.htm、top.htm、left.htm、right.htm、bottom.htm 和 myright.htm。3-4-Frameset.htm 为主索引文件,是本实例的入口文件。

(1) 3-4-Frameset.htm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <frameset rows="15%,75%,10%">
      <frame name="top" src="top.htm" /><--拥有 15%的高度-->
        <frameset cols="25%,75%"><--拥有 75%的高度-->
            <frame name="left" src="left.htm"><--拥有 25%的宽度-->
            <frame name="right" src="right.htm"><--拥有 75%的宽度-->
        </frameset>
      <frame name="bottom" src="bottom.htm" /><--拥有 10%的高度-->
    </frameset>
```

```
</html>
```

(2) top. htm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>freamset——top</title>
    </head>
    <body>
        <div style="text-align:center;color:blue;font-size:50px">(这里是头部)牡丹亭
        </div>
    </body>
</html>
```

(3) left. htm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>freamset——left</title>
    </head>
    <body>
        <div style="color:blue;font-size:30px">(左边)
        <br />
        《牡丹亭》
        <br/>
        作者:汤显祖
        <br/>
        <a target="right" href="right.htm">前两句</a><br/>
        <a target="right" href="myright.htm">后两句</a>
        </div>
    </body>
</html>
```

(4) right. htm

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>freamset——right</title>
  </head>
  <body>
```

```
        <div style="color:blue;font-size:30px"> (右边前两句)
        <br/>
        原来姹紫嫣红开遍
        <br/>
        似这般都付与断井颓垣
        </div>
  </body>
</html>
```

(5) bottom.htm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>freamset--bottom</title>
    </head>

    <body>
        <div style="text-align:center;color:blue;font-size:30px"> (底部)
            《皂罗袍》
        </div>
    </body>
</html>
```

(6) myright.htm

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>freamset——myright</title>
    </head>
    <body>
    <div style="color:blue;font-size:30px"> (右边后两句)
        <br/>
        良辰美景奈何天
        <br/>
        赏心乐事谁家院
    </div>
    </body>
</html>
```

3) 技术要点

(1) frameset 不能和 body 同时使用。

(2) <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">表示 HTML 版本为 4.01。

(3) 凡是 frameset 所在的 HTML 文档,版本不能为 HTML5。

3.5 Bootstrap 布局实例

1. 概述

Bootstrap 是一个用于快速开发 Web 应用程序和网站的前端框架。Bootstrap 基于 HTML、CSS、JavaScript,是由 Twitter 的 Mark Otto 和 Jacob Thornton 开发的。Bootstrap 是 2011 年 8 月在 GitHub 上发布的开源产品。自 Bootstrap 3 起,框架包含了贯穿整个库的移动设备优先的样式。目前,所有的主流浏览器都支持 Bootstrap。Bootstrap 的响应式 CSS 能够自适应于台式机、平板电脑和手机。它为开发人员创建接口提供了一个简洁统一的解决方案,包含了功能强大的内置组件,易于定制,提供了基于 Web 的定制,是开源的。Bootstrap 包的内容如下。

1) 基本结构

Bootstrap 提供了一个带有网格系统、链接样式、背景的基本结构。这将在 Bootstrap 基本结构部分详细讲解。

2) CSS

Bootstrap 自带特性为:全局的 CSS 设置、定义基本的 HTML 元素样式、可扩展的 class 以及一个先进的网格系统。这将在 Bootstrap CSS 部分详细讲解。

3) 组件

Bootstrap 包含十几个可重用的组件,用于创建图像、下拉菜单、导航、警告框、弹出框等。这将在布局组件部分详细讲解。

4) 插件

Bootstrap 包含十几个自定义的 jQuery 插件,读者可以直接包含所有的插件,也可以逐个包含这些插件。

5) 定制

可以定制 Bootstrap 的组件、LESS 变量和 jQuery 插件来得到读者的版本。

2. 布局实例

图 3.8 所示为 Bootstrap 布局实例。

1) 实例分析

在 body 标签中,使用 Bootstrap 提供的 Container 类创建一个块级 div 元素,作为页面放置其他元素的外层容器。在这个外层容器中添加一级标题 H1 元素,作为网页的一级标题。

使用 Bootstrap 提供的 nav 元素创建水平主导航条,由主导航条组织导航菜单项。在 Bootstrap 中,导航条类为 navbar,用于在容器中添加导航条。导航条类默认底色为白

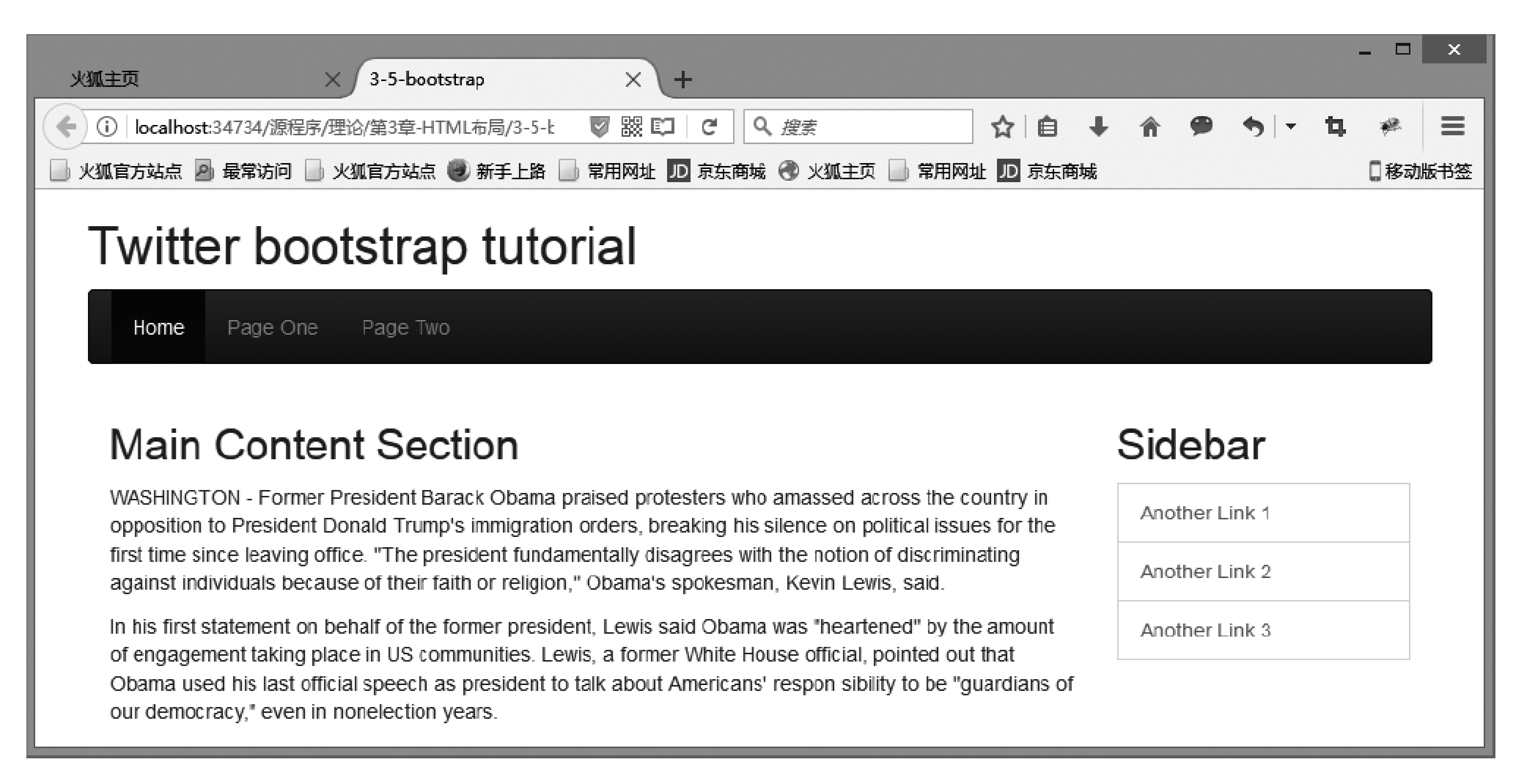


图 3.8 Bootstrap 布局实例(视窗宽≥768px)

色，利用 navbar-inverse 反白类设置反显。导航条由 ul 无序列表元素组织实际导航内容，由 nav 导航类说明一组导航，而由 navbar-nav 类说明导航条中的导航。列表项 li 是实际的导航项目，可以使用 active 来说明当前活动的导航。由 navbar-collapse 类设置，在视窗宽度小于 768px 时将导航变成垂直方向，如图 3.9 所示。

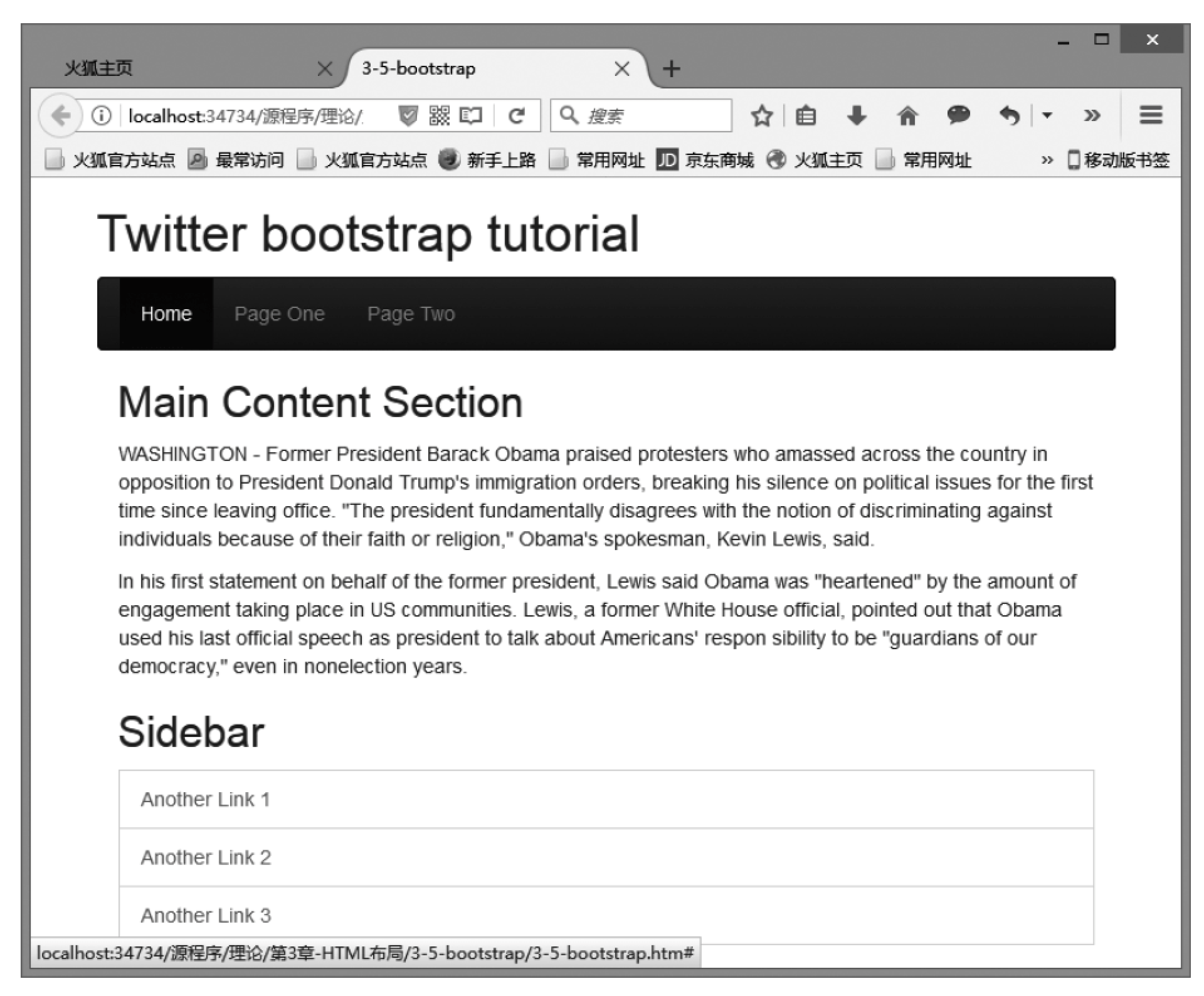


图 3.9 Bootstrap 布局实例(视窗宽<768px)

主要内容部分使用类名为 content 的 div 来布局。这里使用 bootstrap 栅格布局，栅格系统利用 12 列布局，意味着一个页面可以被分隔成 12 个相同列。在侧边栏中添加一些导航内容。普通导航使用 nav 类进行声明，nav-tabs 和 nav-stacked 是导航的外观的 2

个外观类。

2）详细的 HTML 文档

```
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1"/>
  <title>3-5-bootstrap</title>
  <link href="bootstrap-3.3.7-dist/css/bootstrap-theme.min.css"
      rel="stylesheet" type="text/css" />
  <link href="bootstrap-3.3.7-dist/css/bootstrap.min.css"
      rel="stylesheet" type="text/css" />
<style type="text/css">
    .nav-tabs {  border: 1px solid #ddd;margin-bottom:-3px; }
</style>
</head>
<body>
    <div class="container">
        <h1>Twitter bootstrap tutorial</h1>
        <nav class="navbar navbar-inverse">
            <div id="navbar-menu" class="navbar-collapse">
                <ul class="nav navbar-nav">
                    <li class="active"><a href="#">Home</a></li>
                    <li><a href="#">Page One</a></li>
                    <li><a href="#">Page Two</a></li>
                </ul>
            </div>
        </nav>
        <div id="content" class="row-fluid">
            <div class="col-md-9">
                <h2>Main Content Section</h2>
              <pre></pre>
            </div>
            <div class="col-md-3">
                <h2>Sidebar</h2>
                <ul class="nav nav-stacked">
                    <li class="nav-tabs"><a href='#'>Another Link 1</a></li>
                   <li class="nav-tabs"><a href='#'>Another Link 2</a></li>
                   <li class="nav-tabs"><a href='#'>Another Link 3</a></li>
                </ul>
            </div>
        </div>
    </div>
  <script src="jquery-1.3.2.min.js" type="text/javascript"></script>
```

```
<script src="bootstrap-3.3.7-dist/js/bootstrap.min.js"
    type="text/javascript"></script>
</body>
</html>
```

3.6 HTML5 新元素布局实例

HTML5 提供的新语义元素定义了网页的不同部分，如表 3.3 所示。

表 3.3 HTML5 布局新元素

标　签	描　　述	标　签	描　　述
header	定义文档或节的页眉	aside	定义内容之外的内容（比如侧栏）
nav	定义导航链接的容器	footer	定义文档或节的页脚
section	定义文档中的节	details	定义额外的细节
article	定义独立的自包含文章	summary	定义 details 元素的标题

应用 HTML5 新元素进行布局的实例效果如图 3.10 所示。

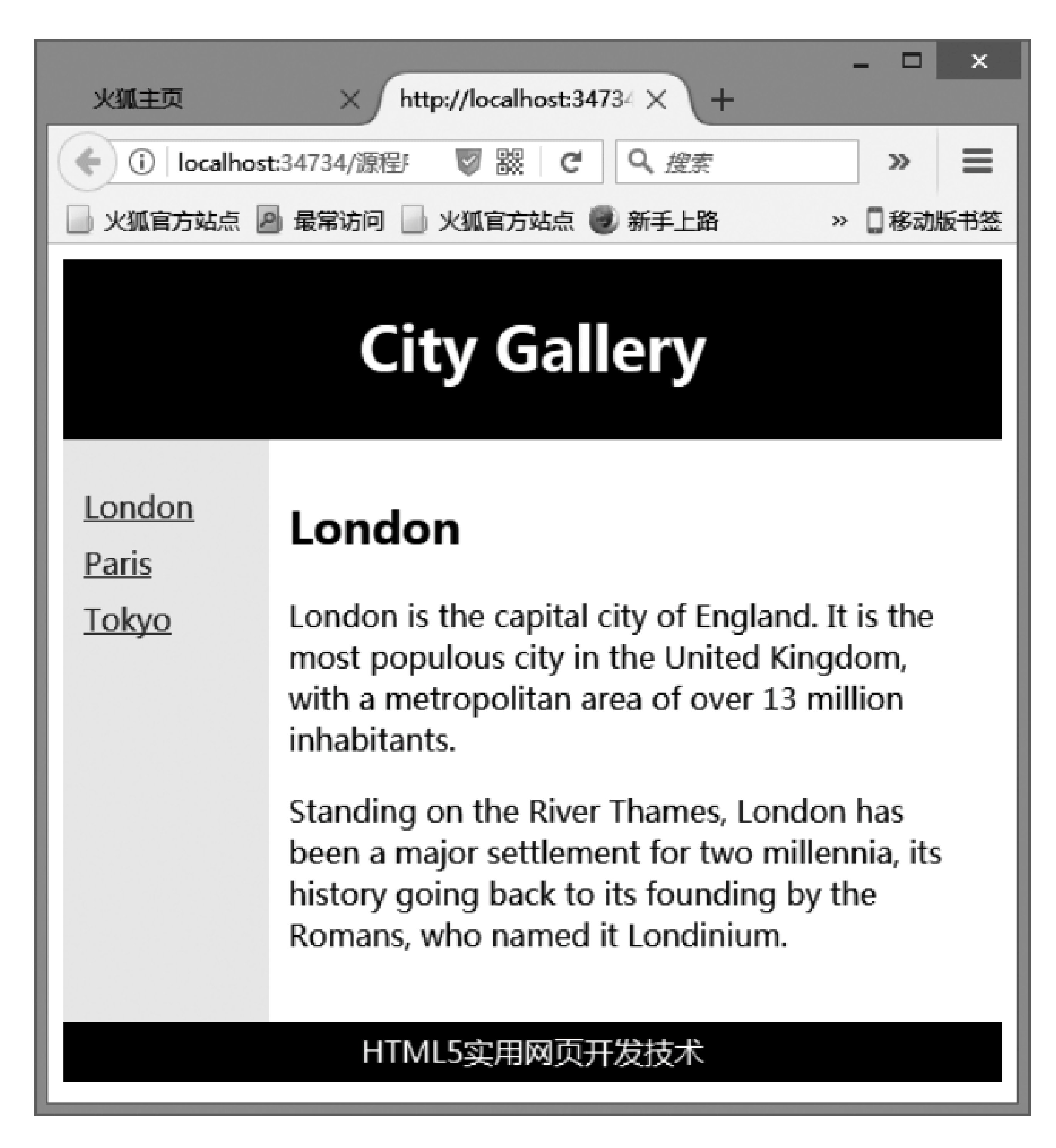


图 3.10 HTML 新元素布局实例

完整的 HTML 文档如下。

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
        <style type="text/css">
                header {
                    background-color:black;
                    color:white;
                    text-align:center;
                    padding:5px;
                }
                nav {
                    line-height:30px;
                    background-color:#eeeeee;
                    height:300px;
                    width:100px;
                    float:left;
                    padding:5px;
                }
                section { width:350px; float:left; padding:10px; }
                footer {
                    background-color:black;
                    color:white;
                    clear:both;
                    text-align:center;
                    padding:5px;
                }
                ul{ padding-left:0px;margin-left:6px;}
                ul>li{list-style-type:none; }
                </style>
            </head>
            <body>
                <header><h1>City Gallery</h1></header>
                <nav>
                    <ul>
                        <li><a href="#">London</a></li>
                        <li><a href="#">Paris</a></li>
                        <li><a href="#">Tokyo</a></li>
                    </ul>
                </nav>
                <section>
                    <h1>London</h1>
                    <p>
```

```
                London is the capital city of England. It is the most populous city in the
                United Kingdom, with a metropolitan area of over 13 million inhabitants.
            </p>
            <p>
                Standing on the River Thames, London has been a major settlement for two
                millennia, its history going back to its founding by the Romans, who named
                it Londinium.
            </p>
        </section>
        <footer>HTML5实用网页开发技术</footer>
    </body>
</html>
```

3.7 实验任务

1. 实验题目

运用 Bootstrap 实现首页流式布局。

2. 程序功能

综合利用所学的标签和 CSS 设计一个画图网页，含横向水平主菜单、纵向树形面板和客户区。横向水平主菜单含有长方形、圆、菱形、正方形、椭圆 5 个菜单项。纵向树形面板含有线型、颜色、动静 3 个选项，线型含有实线、虚线 2 个子项，颜色含有红、绿、蓝 3 个子项，动静含有动和静 2 个子选项。

3. 实验类型

综合设计。

4. 实验要求

(1) 运用 bootstrap 实现流式布局，以适用于 PC、PAD 和手机。
(2) 比例布局合理。
(3) 仅运用内嵌样式。
(4) 横向水平菜单均匀等间距等长分布。
(5) 纵向菜单项均含有图标，横向水平菜单项不用图标。

5. 实验环境

(1) 计算机：PC、内存 8GB、主频 1.8GHz 及以上、硬盘 500GB 及以上。
(2) 操作系统：Windows XP、Windows 7、Windows 8、Windows 10。
(3) 开发环境：Visual Studio 2010 或 Adobe Dreamweaver。

(4) 浏览器：IE9及以上、Chrome、Firefox、Safari、Edge、QQ浏览器等。

6. 实验原理

(1) 图文列表与浮动。
(2) Bootstrap开源控件。

7. 源代码

(1) 辅以必要的注释。
(2) 错落有致，结构清晰，易读性强。

8. 遇到的问题及解决办法

第4章



JavaScript

■ **知识目标**

- 掌握 JavaScript 基础知识
- 理解 JavaScript 的基本特点
- 掌握 JavaScript 的用途

■ **能力目标**

- 能够应用 JavaScript 设计动态 HTML 页面
- 能够准确迅速对 JavaScript 程序错误进行初步判定、跟踪、调试
- 能够对程序异常进行捕获和输出

■ **素质目标**

- 娴熟地单拍跟踪调试、变量监视、断点设置和异常捕获

■ **教学重点**

- 面向对象的 JavaScript 程序设计
- 跟踪调试与异常捕获

■ **教学难点**

- 根据运行错误恰当设置断点,进行单拍跟踪调试

■ **建议学时**

- 理论: C、C++ 、Java 或 C# 开设在先,建议 2 学时,教师讲述要点,学生自学细节
- 实验: 4 学时

4.1 简　介

JavaScript 是一种直译式脚本语言,是一种动态类型、弱类型、基于原型的语言,内置支持类型。它的解释器被称为 JavaScript 引擎,为浏览器的一部分,广泛用于客户端的脚本语言,最早是在 HTML(标准通用标记语言下的一个应用)网页上使用,用来给 HTML 网页增加动态功能。

1. 组成

(1) ECMAScript:描述了该语言的语法和基本对象。
(2) 文档对象模型(DOM):描述处理网页内容的方法和接口。
(3) 浏览器对象模型(BOM):描述与浏览器进行交互的方法和接口。

2. 特点

JavaScript 是一种属于网络的脚本语言,已经被广泛用于 Web 应用开发,常用来为网页添加各式各样的动态功能,为用户提供更流畅美观的浏览效果。通常,JavaScript 脚本是通过嵌入 HTML 中来实现自身的功能的,其基本特点如下:

(1) 解释性:它是一种解释性脚本语言(代码不进行预编译),可以直接嵌入 HTML 页面,也可写成单独的 js 文件(有利于结构和行为的分离)。

(2) 动态性:主要用来向 HTML 页面添加交互行为,JavaScript 是一种采用事件驱动的脚本语言,它不需要经过 Web 服务器就可以对输入、输出、鼠标、键盘、加载、定时等事件进行响应。

(3) 平台无关性:在绝大多数浏览器的支持下,可以在多种平台(如 Windows、Linux、Mac、Android、iOS 等)下运行;不同于服务器端脚本语言,例如 PHP 与 ASP,JavaScript 主要被作为客户端脚本语言,在用户的浏览器上运行,不需要服务器的支持。所以,早期程序员比较青睐 JavaScript,以减少对服务器的负担。

(4) 面向对象:JavaScript 是一种基于对象的脚本语言,它不仅可以创建对象,也能使用现有的对象。

(5) 简单易学:JavaScript 语言中采用的是弱类型的变量类型,对使用的数据类型未做出严格的要求,是基于 Java 基本语句和控制的脚本语言,其设计简单紧凑。

(6) 安全性:浏览器脚本程序安全性低。随着服务器的强壮性增强,安全程序可运行于服务端的脚本,但 JavaScript 仍然以其跨平台、容易上手等优势大行其道。同时,有些特殊功能(如 Ajax)必须依赖 JavaScript 在客户端支持。随着引擎(如 V8)和框架(如 Node. js)的发展及其事件驱动及异步 IO 等特性,JavaScript 逐渐被用来编写服务器端程序。

3. 用途

Javascript 脚本语言同其他语言一样,有其自身的基本数据类型、表达式和算术运算符及程序的基本程序框架。Javascript 提供了 4 种基本的数据类型和 2 种特殊数据类型,用来处理数据和文字。而变量提供存放信息的地方,表达式则可以完成较复杂的信息处理工作。其日常用途如下:

(1) 在 HTML 页面嵌入动态文本。
(2) 对浏览器事件做出响应。
(3) 读写 HTML 元素。
(4) 在数据被提交到服务器之前验证数据。

(5) 检测访客的浏览器信息。

(6) 控制 Cookies,包括创建和修改等。

(7) 基于 Node.js 技术进行服务器端编程。

4. 实现

从理论上讲,可以在 HTML 文档中放入不限数量的脚本。而实际中,过多的脚本处理不当使网页加载变慢。脚本由单独的 script 元素标识,类似于 CSS,可以在所需的 HTML 文档里编写,也可以由外部文件导入。

(1) <script> 标签

HTML 脚本位于<script>与</script> 标签之间。脚本可被放置在<body>或<head>部分中。通常的做法是把函数放入<head>部分中,或者放在页面底部。这样就可以把它们安置到同一处位置,不会干扰页面的内容。<script>和</script>会告诉 JavaScript 在何处开始和结束。<script>和</script>之间的代码行包含了 JavaScript,例如:

```
<script>alert("My First JavaScript");</script>
```

低版本的浏览器实例可能会在<script>标签中使用 type="text/javascript",现已不必如此。JavaScript 是所有现代浏览器以及 HTML5 中的默认脚本语言。

把一个 JavaScript 函数放置到 HTML 页面的<head>部分,如例 4-1-1-1 所示,该函数会在单击按钮时被调用。同样,放在<body>部分如例 4-1-1-2 所示。

例 4-1-1-1 JavaScript 函数放置到 HTML 页面的<head>部分示例。

```
<!DOCTYPE html">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
        <script type="text/javascript">
            function myFunction() {
                document.getElementById("demo").innerHTML="我的第一个 JavaScript 函
                数";
            }
        </script>
    </head>
    <body>
        <h1>我的 Web 页面</h1>
        <p id="demo">一个段落</p>
        <button type="button" onclick="myFunction()">尝试一下</button>
    </body>
</html>
```

例 4-1-1-2 JavaScript 函数放置到 HTML 页面的<body>部分示例。

```
<!DOCTYPE html">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
    </head>
    <body>
        <h1>我的 Web 页面</h1>
        <p id="demo">一个段落</p>
        <button type="button" onclick="myFunction()">尝试一下</button>
    </body>
        <script type="text/javascript">
            function myFunction() {
                document.getElementById("demo").innerHTML="我的第一个 JavaScript 函
                数";
            }
        </script>
</html>
```

可以把脚本保存到外部文件中。外部文件通常是被多个网页引用的脚本。外部 JavaScript 文件的文件扩展名是.js。如需使用外部文件，请在<script>标签的"src"属性中设置该.js 文件，如例 4-1-1-3 所示。

例 4-1-1-3 放置到外部文件的 JavaScript 函数示例。

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
        <script src="例 4-1-1-3.js" type="text/javascript"></script>
    </head>
    <body>
        <h1>我的 Web 页面</h1>
        <p id="demo">一个段落</p>
        <button type="button" onclick="myFunction()">尝试一下</button>
    </body>
</html>
```

外部脚本文件名为：例 4-1-1-3.js，所含内容如下：

```
function myFunction()
{
    document.getElementById("demo").innerHTML="我的第一个 JavaScript 函数";
}
```

注意：外部脚本文件内容不必、也不能包含 <script> 标签。

(2) 简单功能

① 写入 HTML 输出。

```
document.write("<h1>This is a heading</h1>");
document.write("<p>This is a paragraph</p>");
```

② 对事件做出反应。

```
<button type="button" onclick="alert('Welcome!')">单击这里</button>
```

③ 改变 HTML 内容

```
x=document.getElementById("demo")        //查找元素
x.innerHTML="Hello JavaScript";           //改变内容
```

④ 改变 HTML 样式。

```
x=document.getElementById("demo")        //找到元素
x.style.color="#ff0000";                  //改变样式
```

5. 输出

JavaScript 没有任何打印或者输出的函数。JavaScript 可以通过如下 4 种不同方式来输出数据:

- 使用 window.alert()弹出警告框。
- 使用 document.write()方法,将内容写到 HTML 文档中。
- 使用 innerHTML 写入 HTML 元素。
- 使用 console.log()写入浏览器的控制台。

(1) 使用 window.alert()

可以弹出警告框来显示数据,如例 4-1-1-4 所示。

例 4-1-1-4 使用 window.alert()弹出警告框示例。

```
<!DOCTYPE html>
<html>
    <head></head>
    <body>
        <h1>我的第一个页面</h1>
        <p>我的第一个段落。</p>
        <script type="text/javascript">
            window.alert(5+6);
        </script>
    </body>
</html>
```

(2) 操作 HTML 元素

如需从 JavaScript 访问某个 HTML 元素,可以使用 document.getElementById(id)方法,操纵由"id"属性标识的 HTML 元素,从 innerHTML 来获取,或向 innerHTML 插

入元素内容，如例 4-1-1-5 所示。

例 4-1-1-5 JavaScript 操作 HTML 元素示例。

```
<!DOCTYPE html>
<html>
    <head><title></title></head>
    <body>
        <h1>我的第一个 Web 页面</h1>
        <p id="demo">我的第一个段落</p>
        <script type="text/javascript">
            document.getElementById("demo").innerHTML="段落已修改。";
        </script>
    </body>
</html>
```

以上 JavaScript 语句(在 <script>标签中)可以在 Web 浏览器中执行。document.getElementById("demo")是使用 id 属性来查找 HTML 元素的 JavaScript 代码。innerHTML="段落已修改。"是用于修改元素的 HTML 内容(innerHTML)的 JavaScript 代码。

(3) 写到 HTML 文档

可以将 JavaScript 直接写在 HTML 文档中，如例 4-1-1-6 所示。

例 4-1-1-6 JavaScript 修改 HTML 文档示例。

```
<!DOCTYPE html>
<html>
    <head><title></title></head>
    <body>
    <h1>我的第一个 Web 页面</h1>
    <p>我的第一个段落。</p>
    <script type="text/javascript">
        document.write(Date());
    </script>
    </body>
</html>
```

如果在文档完成加载后执行 document.write，整个 HTML 页面将被覆盖，如例 4-1-1-7 所示。

例 4-1-1-7 被例 4-1-1-6 修改后的 HTML 文档。

```
<!DOCTYPE html>
<html>
    <head><title></title></head>
    <body>
        <h1>我的第一个 Web 页面</h1>
        <p>我的第一个段落。</p>
```

```
        <button onclick="myFunction()">点我</button>
        <script type="text/javascript"> function myFunction() {   document.write(Date
        ());   }</script>
    </body>
</html>
```

(4) 写到控制台

对于支持调试的浏览器,可以使用 console.log()方法在浏览器中显示 JavaScript 值。

```
<!DOCTYPE html>
<html>
    <head><title></title></head>
    <body>
        <h1>我的第一个 Web 页面</h1>
        <script type="text/javascript">
            a=5;
            b=6;
            c=a+b;
            console.log(c);
        </script>
    </body>
</html>
```

浏览器显示后,按 F12 键启用开发人员工具,进入调试模式,在调试窗口中单击"控制台"菜单项,选中"日志"标签卡,如图 4.1 所示。



图 4.1 浏览器控制台输出数据

4.2 理论基础

4.2.1 语法

JavaScript 是一种程序语言。语法规则定义了语言结构。JavaScript 是一个脚本语言。它是一种轻量级但功能强大的编程语言。

1. JavaScript 字面量

在编程语言中，一个字面量是一个常量，如 3.14。数字字面量可以是整数或小数，或者是科学计数(e)。如 3.14、1001 或 123e5。字符串字面量使用单引号或双引号括起来，如"John Doe"、'John Doe'。表达式字面量用于计算，如 5+6、5 * 10。数组字面量定义一个数组，用方括号括起来，如[40,100,1,5,25,10]。对象字面量定义一个对象，用花括号把属性名值对括起来，属性名值对用逗号分隔，属性名和值用冒号匹配，如{firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"}。函数字面量定义一个函数，如 function myFunction(a, b) { return a * b;}。

2. JavaScript 变量

在编程语言中，变量用于存储数据值。JavaScript 使用关键字 var 来定义变量，使用等号来为变量赋值，如：

```
var x, length;
x=5;
length=6;
```

变量可以通过变量名访问。在指令式语言中，变量通常是可变的。字面量是一个恒定的值。

3. JavaScript 操作符

JavaScript 使用算术运算符来计算值，如(5+6) * 10。JavaScript 使用赋值运算符给变量赋值，如：

```
x=5;
y=6;
z=(x+y) * 10;
```

JavaScript 语言有多种类型的运算符，包括赋值、算术、位、条件、比较及逻辑运算符。

4. JavaScript 语句

在 HTML 中，JavaScript 语句向浏览器发出命令。语句用分号分隔，如：

```
x=5+6;
```

```
y=x * 10;
```

5. JavaScript 关键字

JavaScript 关键字用于标识要执行的操作。var 关键字告诉浏览器创建一个新的变量，如：

```
var x=5+6;
var y=x * 10;
```

和其他任何编程语言一样，JavaScript 保留了一些关键字，为自己所用。JavaScript 同样保留了一些关键字，这些关键字在当前的语言版本中并没有使用，但在以后的 JavaScript 扩展中会用到。JavaScript 关键字必须以字母、下画线(_)或美元符($)开始。后续的字符可以是字母、数字、下画线或美元符(数字是不允许作为首字符出现的，以便 JavaScript 可以轻易区分关键字和数字)。

以下是 JavaScript 中最重要的保留字(按字母顺序)。

abstract、else、instanceof、super、boolean、enum、int、switch、break、export、interface、synchronized、byte、extends、let、this、case、false、long、throw 等。

6. JavaScript 注释

不是所有的 JavaScript 语句都是“命令”。双斜杠//后的内容将会被浏览器忽略，例如：

```
//我不会执行
```

7. JavaScript 数据类型

JavaScript 有多种数据类型，包括数字、字符串、数组、对象等。在编程语言中，数据类型是一个非常重要的内容。为了可以操作变量，了解数据类型的概念非常重要。如果没有使用数据类型，以下实例将无法执行：

```
16  +"Hello"
```

16 加上"Hello"会输出："16Hello"。

8. JavaScript 函数

JavaScript 语句可以写在函数内，函数可以重复引用。引用一个函数等同于调用函数(执行函数内的语句)。函数定义如下：

```
function myFunction(a, b) {
    return a * b;                    //返回 a 乘于 b 的结果
}
```

9. JavaScript 对大小写敏感

JavaScript 对大小写是敏感的。编写 JavaScript 语句时，请留意是否关闭大小写切

换键。函数 getElementById 与 getElementbyID 不同。同样,变量 myVariable 与 MyVariable 也不同。

4.2.2 语句

JavaScript 语句是向浏览器发出的命令。语句的作用是告诉浏览器该做什么。下面的 JavaScript 语句向 id="demo"的 HTML 元素输出文本"你好 Dolly"。

```
document.getElementById("demo").innerHTML="你好 Dolly";
```

1. 分号

分号用于分隔 JavaScript 语句。通常,每条可执行的语句结尾添加分号。分号的另一用处是在一行中编写多条语句。例如:

```
a=5;
b=6;
c=a+b;
```

以上实例也可以写成

```
a=5; b=6; c=a+b;
```

在 JavaScript 中,用分号来结束语句是可选的。

2. JavaScript 代码

JavaScript 代码是 JavaScript 语句的序列。浏览器按照编写顺序依次执行每条语句。以下 2 个语句顺序执行。

```
document.getElementById("demo").innerHTML="你好 Dolly";
document.getElementById("myDIV").innerHTML="你最近怎么样?";
```

3. JavaScript 代码块

JavaScript 可以分批地组合起来。代码块以左花括号开始,以右花括号结束。代码块的作用是一并地执行语句序列。以下所示代码块作为一个函数体运行。

```
function myFunction()
{
    document.getElementById("demo").innerHTML="你好 Dolly";
    document.getElementById("myDIV").innerHTML="你最近怎么样?";
}
```

4. JavaScript 语句标识符

JavaScript 语句通常以一个语句标识符为开始,并执行该语句。语句标识符是保留关键字,不能作为变量名、函数名、对象名或数组名使用。

5. 空格

JavaScript 会忽略多余的空格。可以向脚本添加空格,以提高语句可读性。下面的两行代码是等效的:

```
var person="Hege";
var person = "Hege";
```

6. 对代码行进行折行

可以在文本字符串中使用反斜杠,对代码行进行换行。下面的例子会正确地显示。

```
document.write("你好 \
世界!");
```

4.2.3 变量

变量是用于存储信息的“容器”。例如:

```
var x=5;
var y=6;
var z=x+y;
```

变量运算就像代数运算那样,使用字母(比如 x)来保存值(比如 5)。通过上面的表达式 z=x+y,能够计算出 z 的值为 11。在 JavaScript 中,这些字母被称为变量。变量可以使用短名称(比如 x 和 y),也可以使用描述性更好的名称(比如 age、sum、totalvolume)。变量必须以字母开头,也能以 $ 和_符号开头。变量名称对大小写敏感(y 和 Y 是不同的变量)。

JavaScript 变量还能保存其他数据类型,比如文本值(name="Bill Gates")。在 JavaScript 中,类似"Bill Gates"这样一条文本被称为字符串。当向变量分配文本值时,应该用双引号或单引号包围这个值。当向变量赋的值是数值时,不要使用引号。如果用引号包围数值,该值会被作为文本来处理。例如:

```
var pi=3.14;
var person="John Doe";
var answer='Yes I am!';
```

在 JavaScript 中创建变量,通常称为“声明”变量。使用 var 关键词来声明变量,例如:

```
var carname;
```

变量声明之后,该变量是空的(它没有值)。如需向变量赋值,使用等号,例如:

```
carname="Hello";
```

不过,也可以在声明变量时对其赋值,如

```
var carname="Hello";
```

下面的例子创建了名为 carname 的变量,并向其赋值"Hello",然后把它放入 id="demo"的 HTML 段落中。

```
<p id="demo"></p>
var carname="Hello";
document.getElementById("demo").innerHTML=carname;
```

一个好的编程习惯是,在代码开始处统一对需要的变量进行声明。可以在一条语句中声明很多变量。该语句以 var 开头,并使用逗号分隔变量。

```
var lastname="Doe", age=30, job="carpenter";
```

声明也可横跨多行,例如:

```
var lastname="Doe",
age=30,
job="carpenter";
```

计算机程序中经常会声明无值的变量。未使用值来声明的变量,其值实际是 undefined。

执行过以下语句后,变量 carname 的值将是 undefined。

```
var carname;
```

如果重新声明 JavaScript 变量,该变量的值不会丢失。以下两条语句执行后,变量 carname 的值依然是"Hello"。

```
var carname="Hello";
var carname;
```

可以通过 JavaScript 变量来做算数,使用的是=和+这类运算符,例如:

```
y=5;
x=y+2;
```

4.2.4 数据类型

JavaScript 的数据类型有字符串、数字、布尔、数组、对象、空、未定义。JavaScript 变量拥有动态类型,这意味着相同的变量可用做不同的类型,例如:

```
var x;                //x 为 undefined
var x=5;              //现在 x 为数字
var x="John";         //现在 x 为字符串
```

1. 字符串

字符串是存储字符(比如"Bill Gates")的变量。字符串可以是引号中的任意文本。

可以使用单引号或双引号，例如：

```
var carname="Hello HTML5 JS";
var carname='Hello HTML5 JS';
```

可以在字符串中使用引号，只要与包围字符串的引号不匹配即可，例如：

```
var answer="It's all right";
var answer="He is called 'Johnny'";
var answer='He is called "Johnny"';
```

2. 数值

JavaScript 只有一种数字类型。数字可以带小数点，也可以不带，例如：

```
var x1=34.00;        //使用小数点来写
var x2=34;           ////不使用小数点来写
```

极大或极小的数字可以通过科学(指数)计数法来书写，例如：

```
var y=123e5;         //12300000
var z=123e-5;        //0.00123
```

3. 布尔

布尔(逻辑)只能有两个值：true 或 false，例如：

```
var x=true;
var y=false;
```

布尔常用在条件测试中。

4. 数组

数组可以用 new Array()赋值语句申请，而后按分量赋值，例如：

```
var cars=new Array();
cars[0]="Saab";
cars[1]="Hello";
cars[2]="BMW";
```

或者

```
var cars=new Array("Saab","Hello","BMW");
```

或者

```
var cars=["Saab","Hello","BMW"];
```

数组下标是基于 0 的，所以第一个项目是[0]，第二个是[1]，以此类推。

5. 对象

对象由花括号分隔。在括号内部,对象的属性以名称和值对的形式(name :value)来定义。属性由逗号分隔,例如:

```
var person={firstname:"John", lastname:"Doe", id:5566};
```

上面例子中的对象(person)有3个属性:firstname、lastname以及id。

空格和折行无关紧要。声明可横跨多行:

```
var person={
    firstname : "John",
    lastname : "Doe",
    id: 5566
};
```

对象属性有如下两种寻址方式:

```
name=person.lastname;
name=person["lastname"];
```

6. Undefined 和 Null

Undefined这个值表示变量不含有值。可以通过将变量的值设置为null来清空变量。例如:

```
cars=null;
person=null;
```

7. 声明变量类型

声明新变量时,可以使用关键词new来声明其类型,例如:

```
var carname=new String;
var x=new Number;
var y=new Boolean;
var cars=new Array;
var person=new Object;
```

JavaScript 变量均为对象。声明一个变量后,就创建了一个新的对象。

4.2.5 对象

JavaScript 对象是拥有属性和方法的数据。真实生活中的对象有属性和方法。如一辆汽车是一个对象,有重量和颜色等属性,有启动停止等方法,如表4.1所示。

所有汽车都有这些属性,但每款车的属性值都不尽相同。所有汽车都拥有这些方法,但它们被执行的时间都不尽相同。

表 4.1 汽车的属性和方法

属　　性	方　法	属　　性	方　法
car. name＝Fiat	car. start()	car. weight＝850kg	car. brake()
car. model＝500	car. drive()	car. color＝white	car. stop()

在 JavaScript 中，几乎所有的事物都是对象。以下代码为变量 car 设置值为 Fiat，例如：

```
var car="Fiat";
```

对象也是一个变量，但对象可以包含多个值(多个变量)，例如：

```
var car={type:"Fiat", model:500, color:"white"};
```

在上例中，3 个值被("Fiat", 500, "white")赋予变量 car。

1. 定义

因此，JavaScript 对象是变量的容器。可以使用字符来定义和创建 JavaScript 对象，例如：

```
var person={firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

定义 JavaScript 对象可以跨越多行，空格跟换行不是必需的，例如：

```
var person={
    firstName:"John",
    lastName:"Doe",
    age:50,
    eyeColor:"blue"
};
```

可以说，JavaScript 对象是变量的容器。进一步说，JavaScript 对象是键值对的容器。键值对的通常写法为 name ：value(键与值以冒号分隔)。键值对在 JavaScript 对象中通常称为对象属性。

2. 访问对象属性

可以通过如下两种方式访问对象属性：

```
person.lastName;
person["lastName"];
```

3. 访问对象方法

对象的方法定义了一个函数，并作为对象的属性存储。对象方法通过添加()调用(作为一个函数)。例如：

```
name=person.fullName();
```

如果要访问 person 对象的 fullName 属性，它将作为一个定义函数的字符串返回，如下所示：

```
name=person.fullName;
```

可以使用以下语法创建对象方法：

```
methodName : function() { code lines }
```

可以使用以下语法访问对象方法：

```
objectName.methodName()
```

通常，fullName()是作为 person 对象的一个方法，fullName 是作为一个属性。

有多种方式可以创建、使用和修改 JavaScript 对象。同样也有多种方式创建、使用和修改属性和方法。

4.2.6 函数

函数是由事件驱动，或者当函数被调用时执行的可重复使用的代码块，如例 4-2-6-1 所示。

例 4-2-6-1 一个简单的单击事件。

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
        <script type="text/javascript">
            function myFunction() {
                alert("Hello World!");
            }
        </script>
    </head>
    <body>
        <button onclick="myFunction()">Try it</button>
    </body>
</html>
```

1. 语法

函数就是包裹在花括号中的代码块，前面使用了关键词 function，如下所示：

```
function functionname(){  执行代码  }
```

调用该函数时，会执行函数内的代码。可以在某事件发生时直接调用函数(比如当用户单击按钮时)，并且可由 JavaScript 在任何位置调用。JavaScript 对大小写敏感。关

键词 function 必须小写，并且必须以与函数名称相同的大小写来调用函数。

2. 参数

调用函数时，可以向其传递值，这些值被称为参数。这些参数可以在函数中使用。可以发送任意多的参数，形参置于()内，形参间用逗号分隔，如下所示：

```
myFunction(argument1, argument2)
```

声明函数时，把参数作为变量来声明，如下所示：

```
function myFunction(var1,var2){ 代码 }
```

变量和参数必须以一致的顺序出现。第一个变量就是第一个被传递的参数给定的值，以此类推，如例 4-2-6-2 所示。

例 4-2-6-2 带参数的函数调用示例。

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
        <script type="text/javascript">
            function myFunction(name, job) {
                alert("Welcome "+name+", the "+job);
            }
        </script>
    </head>
    <body>
        <p>单击这个按钮,来调用带参数的函数。</p>
        <button onclick="myFunction('Harry Potter','Wizard')">单击这里
            </button>
    </body>
</html>
```

上面的函数在按钮被单击时会提示"Welcome Harry Potter, the Wizard"。函数很灵活，可以使用不同的参数调用该函数，这样就会给出不同的消息。

3. 返回值

有时需要函数将值返回给调用者，使用 return 语句就可以实现。使用 return 语句时，函数会停止执行，并返回指定的值。如下所示：

```
function myFunction()
{
    var x=5;
    return x;
}
```

上面的函数返回值5后,myFunction函数停止运行,返回到myFunction函数的调用处继续执行代码。函数调用将被返回值取代,如下所示:

```
var myVar=myFunction();
```

myVar变量的值是5,也就是函数myFunction()所返回的值。即使不把它保存为变量,也可以使用返回值,例如:

```
document.getElementById("demo").innerHTML=myFunction();
```

"demo"元素的innerHTML将成为5,也就是函数myFunction()所返回的值。可以基于传递到函数中的参数计算返回值,例如:

```
function myFunction(a,b)
{
    return a * b;
}
document.getElementById("demo").innerHTML=myFunction(4,3);
```

"demo"元素的innerHTML将是:

```
12
```

在仅希望退出函数时,也可以使用return语句。返回值是可选的,例如:

```
function myFunction(a,b)
{
    if (a>b)
    {
        return;
    }
    x=a+b
}
```

如果a大于b,则上面的代码将退出函数,并不会计算a和b的总和。

4. 局部量

JavaScript函数内部声明的变量是局部变量,所以只能在函数内部访问它。可以在不同的函数中使用名称相同的局部变量,因为只有声明过该变量的函数才能识别出该变量。只要函数运行完毕,本地局部变量就会被删除。

5. 全局量

在函数外声明的变量是全局变量,网页上的所有脚本和函数都能访问它。

6. 变量的生存期

变量的生命期从它们被声明的时间开始。局部变量会在函数运行以后被删除。全

局变量会在页面关闭后被删除。如果把值赋给尚未声明的变量，该变量将被自动作为全局变量声明。例如：

```
carname="Hello";
```

将声明一个全局变量 carname，即使它在函数内执行，carname 也是全局变量。

4.2.7 字符串

JavaScript 字符串用于存储和处理文本。字符串可以存储一系列字符，如"John Doe"。字符串可以是插入引号中的任何字符。使用单引号或双引号将字符串括起来，例如：

```
var carname="Hello HTML5 JS";
var carname='Hello HTML5 JS';
```

可以使用索引位置来访问字符串中的每个字符，例如：

```
var character=carname[7];
```

字符串的索引从 0 开始，这意味着第一个字符索引值为[0]，第二个为[1]，以此类推。

可以在字符串中使用引号，只要字符串中的引号不与字符串的引号相同即可，例如：

```
var answer="It's all right";
var answer="He is called 'Johnny'";
var answer='He is called "Johnny"';
```

也可以在字符串中添加转义字符来使用引号，例如：

```
var x='It\'s all right';
var y="He is called \"Johnny\"";
```

1. 字符串长度

可以使用内置属性 length 来计算字符串的长度，例如：

```
var txt="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln=txt.length;
```

2. 特殊字符

在 JavaScript 中，如果字符串写在单引号或双引号中，会导致 JavaScript 无法解析，例如：

```
"We are the so-called "Vikings" from the north."
```

字符串"We are the so-called"被截断。使用反斜杠(\)来转义"Vikings"字符串中的双引号，如下所示：

```
"We are the so-called \"Vikings\" from the north."
```

反斜杠是一个转义字符。转义字符将特殊字符转换为字符串字符。转义字符(\)可以用于转义撇号、换行、引号等其他特殊字符。表 4.2 列举了在字符串中可以使用转义字符转义的特殊字符。

表 4.2 特殊字符转回表

代码	输　出	代码	输　出
\'	单引号	\r	回车
\"	双引号	\t	tab(制表符)
\\	反斜杠	\b	退格符
\n	换行	\f	换页符

3. 属性

字符串属性及其描述如表 4.3 所示。

表 4.3 字符串属性及其描述

属　性	描　　述
constructor	返回创建字符串属性的函数
length	返回字符串的长度
prototype	允许向对象添加属性和方法

4. 方法

字符串方法及其描述如表 4.4 所示。

表 4.4 字符串方法及其描述

方　　法	描　　述
charAt()	返回指定索引位置的字符
charCodeAt()	返回指定索引位置字符的 Unicode 值
concat()	连接两个或多个字符串,返回连接后的字符串
fromCharCode()	将 Unicode 转换为字符串
indexOf()	返回字符串中检索指定字符第一次出现的位置
lastIndexOf()	返回字符串中检索指定字符最后一次出现的位置
localeCompare()	用本地特定的顺序来比较两个字符串
match()	找到一个或多个正则表达式的匹配
replace()	替换与正则表达式匹配的子串

续表

方　法	描　述
search()	检索与正则表达式相匹配的值
slice()	提取字符串的片段，并在新的字符串中返回被提取的部分
split()	把字符串分割为子字符串数组
substr()	从起始索引号提取字符串中指定数目的字符
substring()	提取字符串中两个指定的索引号之间的字符
toLocaleLowerCase()	根据主机的语言环境把字符串转换为小写，只有几种语言（如土耳其语）具有地方特有的大小写映射
toLocaleUpperCase()	根据主机的语言环境把字符串转换为大写，只有几种语言（如土耳其语）具有地方特有的大小写映射
toLowerCase()	把字符串转换为小写
toString()	返回字符串对象值
toUpperCase()	把字符串转换为大写
trim()	移除字符串首尾空白
valueOf()	返回某个字符串对象的原始值

4.2.8 运算符

JavaScript 运算符有算数运算符、赋值运算符、关系运算符、逻辑运算符、条件。

1. 算术运算符

表 4.5 给了算术运算符的解释。

表 4.5　算术运算符的解释

运算符	描　述	例　子	x	y
＋	加法	x＝y＋2	7	5
－	减法	x＝y－2	3	5
*	乘法	x＝y * 2	10	5
/	除法	x＝y/2	2.5	5
%	取模（余数）	x＝y%2	1	5
＋＋	自增	x＝＋＋y	6	6
		x＝y＋＋	5	6
－－	自减	x＝y－－	4	4
		x＝－－y	5	4

2. 赋值运算符

赋值运算符用于给 JavaScript 变量赋值。给定 x=10 和 y=5，表 4.6 给出了赋值运算符的解释。

表 4.6　赋值运算符的解释

运算符	例　子	等同于	运算结果
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
*=	x*=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

3. 用于字符串的+运算符

+运算符用于把文本值或字符串变量加起来(连接起来)。如需把两个或多个字符串变量连接起来，就使用+运算符。例如：

```
txt1="What a very";
txt2="nice day";
txt3=txt1+txt2;
```

txt3 的运算结果如下：

```
What a verynice day
```

如在两个字符串之间增加空格，需要把空格插入一个字符串之中，如下所示：

```
txt1="What a very ";
txt2="nice day";
txt3=txt1+txt2;
```

在以上语句执行后，变量 txt3 包含的值如下：

```
What a very nice day
```

或者把空格插入表达式中：

```
txt1="What a very";
txt2="nice day";
txt3=txt1+" "+txt2;
```

以上语句执行后，变量 txt3 包含的值如下：

```
What a very nice day
```

4. 对字符串和数字进行加法运算

两个数字相加，返回数字相加的和，如果数字与字符串相加，返回字符串。例如：

```
x=5+5;
y="5"+5;
z="Hello"+5;
```

x、y 和 z 输出的结果如下：

```
10
55
Hello5
```

5. 比较运算符和逻辑运算符

比较运算和逻辑运算用于测试 true 或者 false。比较运算符在逻辑语句中使用，以测定变量或值是否相等。设 x=5，表 4-7 给出了比较运算符的解释。

表 4.7 比较运算符的解释

运算符	描　述	比　较	返回值
==	等于	x==8	false
		x==5	true
===	绝对等于(值和类型均相等)	x==="5"	false
		x===5	true
!=	不等于	x!=8	true
!==	不绝对等于(值或类型均不相等)	x!=="5"	true
		x!==5	false
>	大于	x>8	false
<	小于	x<8	true
>=	大于或等于	x>=8	false
<=	小于或等于	x<=8	true

可以在条件语句中使用比较运算符，对值进行比较，然后根据结果采取行动：

```
if (age<18) x="Too young";
```

逻辑运算符用于测定变量或值之间的逻辑。给定 x=6 以及 y=3，表 4.8 给出了逻辑运算符的解释。

表 4.8 逻辑运算符的解释

运算符	描 述	例 子
&&	and	(x<10 && y>1) 为 true
||	or	(x==5||y==5) 为 false
!	not	!(x==y) 为 true

6. 条件运算符

JavaScript 还包含了基于某些条件对变量进行赋值的条件运算符。语法如下：

```
variablename=(condition)?value1:value2
```

例如：如果变量 age 中的值小于 18，则向变量 voteable 赋值 "年龄太小"，否则赋值 "年龄已达到"。

```
voteable=(age<18)?"年龄太小":"年龄已达到";
```

4.2.9 条件语句

条件语句用于基于不同条件来执行不同的动作。可以在代码中使用如下条件语句完成该任务：

① if 语句：只有当指定条件为 true 时，使用该语句执行代码。
② if...else 语句：当条件为 true 时执行代码，当条件为 false 时执行其他代码。
③ if...else if.... else 语句：使用该语句，选择多个代码块之一来执行。
④ switch 语句：使用该语句选择多个代码块之一来执行。

1. if 语句

只有当指定条件为 true 时，该语句才会执行代码。语法如下：

```
if (condition)
{
    当条件为 true 时执行的代码
}
```

如果时间小于 20:00，生成问候"Good Day"，如下所示。

```
if (time<20)
{
    x="Good Day";
}
```

这个 if 语句的语法中没有...else...，即只有在指定条件为 true 时才执行代码。

2. if...else 语句

条件为 true 时执行代码，条件为 false 时执行其他代码。语法如下：

```
if (condition)
{
    当条件为 true 时执行的代码
}
else
{
    当条件不为 true 时执行的代码
}
```

如果时间小于 20：00，生成问候"Good Day"，否则生成问候"Good Evening"，如下所示。

```
if (time<20)
{
    x="Good Day";
}
else
{
    x="Good Evening";
}
```

3. if...else if...else 语句

选择多个代码块之一来执行。语法如下：

```
if (condition1)
{
    当条件 1 为 true 时执行的代码
}
else if (condition2)
{
    当条件 2 为 true 时执行的代码
}
else
{
    当条件 1 和条件 2 都不为 true 时执行的代码
}
```

如果时间小于 10：00，则生成问候 "Good Morning"，如果时间大于 10：00 小于 20：00，则生成问候"Good Day"，否则生成问候"Good Evening"，如下所示。

```
if (time<10)
```

```
{
    document.write("<b>Good Morning</b>");
}
else if (time>=10 && time<20)
{
    document.write("<b>Good Day</b>");
}
else
{
    document.write("<b>Good Evening!</b>");
}
```

4. switch 语句

用于基于不同条件执行不同的动作。使用 switch 语句可以选择要执行的多个代码块之一。

（1）语法

```
switch(n)
{
    case 1:
        执行代码块 1
        break;
    case 2:
        执行代码块 2
        break;
    default:
        与 case 1 和 case 2 不同时执行的代码
}
```

工作原理：首先设置表达式 n(通常是一个变量)。随后表达式的值会与结构中的每个 case 的值作比较。如果有匹配项，则与该 case 关联的代码块会被执行。使用 break 可以阻止代码自动地向下一个 case 运行。

显示今天的星期名称(设 Sunday＝0、Monday＝1、Tuesday＝2，以此类推)，如下所示。

```
var d=new Date().getDay();
switch (d)
{
    case 0:x="今天是星期日"; break;
    case 1:x="今天是星期一"; break;
    case 2:x="今天是星期二"; break;
    case 3:x="今天是星期三"; break;
    case 4:x="今天是星期四"; break;
```

```
    case 5:x="今天是星期五"; break;
    case 6:x="今天是星期六"; break;
}
```

(2) default 关键词

使用 default 关键词来规定匹配不存在时所执行的程序。

如果今天不是星期六或星期日,则会输出默认的消息,如下所示。

```
var d=new Date().getDay();
switch (d)
{
    case 6:x="今天是星期六";    break;
    case 0:x="今天是星期日";    break;
    default:    x="期待周末";
}
document.getElementById("demo").innerHTML=x;
```

4.2.10 循环语句

JavaScript 支持 4 种不同类型的循环。

① for:循环代码块执行一定的次数。

② for/in:循环遍历对象数组。

③ while:指定的条件为 true 时循环指定的代码块。

④ do/while:同样,当指定的条件为 true 时循环指定的代码块。

1. for 循环

for 循环是经常用到的循环工具,语法如下:

```
for (语句 1; 语句 2;语句 3)
{
    被执行的代码块
}
```

语句 1 在被执行的代码块前执行。语句 2 定义运行反复执行代码块需要满足的条件。语句 3 在每执行一次代码块之后执行。实例如下。

```
for (var i=0; i<5; i++){ x=x+"该数字为 "+i+"<br>"; }
```

在上面的例子中,语句 1,即 var i=0,在循环开始之前设置变量;语句 2,即 i<5,定义循环运行的条件;语句 3,即 i++,在每次代码块已被执行后增加一个值。通常使用语句 1 初始化循环中所用的变量。语句 1 是可选的,也就是说,不使用语句 1 也可以。在语句 1 中可以初始化任意(或者多个)值,如下面的实例所示。

```
for (var i=0,len=cars.length; i<len; i++) { document.write(cars[i]+"<br>"); }
```

语句 1 也可省略(比如在循环开始前已经设置了值时),例如:

```
var i=2,len=cars.length;
for (; i<len; i++) { document.write(cars[i]+"<br>"); }
```

语句 2 通常用于评估初始变量的条件,同样是可选的。如果语句 2 返回 true,则循环再次开始,如果返回 false,则循环将结束。

如果省略了语句 2,必须在循环内提供 break。否则循环就无法停下来。这样有可能出现死循环,使浏览器崩溃。语句 3 一般会增加初始变量的值,也是可选的。语句 3 有多种用法。增量可以是负数(i——),或者更大(i=i+15)。语句 3 也可以省略(如当循环内部有相应的代码时),语句如下:

```
var i=0,len=cars.length;
for (; i<len; ){ document.write(cars[i]+"<br>"); i++; }
```

2. for/in 循环

for/in 语句循环遍历对象数组,例如:

```
var person=[{fname:"John",lname:"Doe", age:25}, {fname:"Tom",lname: "ok", age: 22}];
var txt=0;
for (var x in person) {
    txt=txt+person[x].age;
}
```

3. while 循环

只要指定条件为 true,循环就可以一直执行代码块。语法如下:

```
while (条件)
{
    需要执行的代码
}
```

例如:

```
var i=0;
while (i<5)
{
    x=x+"The number is "+i+"<br>";
    i++;
}
```

如果忘记增加条件中所用变量的值,该循环永远不会结束,这可能导致浏览器崩溃。

4. do/while 循环

do/while 循环是 while 循环的变体。该循环会在检查条件是否为真前执行一次代码

块，如果条件为真的话，就会重复这个循环。语法如下：

```
do
{
    需要执行的代码
}
while (条件);
```

下面的例子使用 do/while 循环。该循环至少会执行 1 次，即使条件为 false，也会执行 1 次，因为代码块会在条件被测试前执行。

```
var i=0;
do
{
    x=x+"The number is "+i+"<br>";
    i++;
}
while (i<5);
```

注意：漏掉增加控制条件变量的值，程序会陷入死循环。

5. break 和 continue 语句

break 语句用于跳出循环。continue 用于跳过循环中的一个迭代。continue 语句跳出循环后会继续执行该循环之后的代码，例如：

```
for (i=0;i<10;i++)
{
    if (i==3)
    {
        break;
    }
    x=x+"The number is "+i+"<br>";
}
```

由于这个 if 语句只有 1 行代码，因此花括号可以省略。

continue 语句中断循环中的迭代，如果出现了指定条件，会继续循环中的下一个迭代。例如：

```
for (i=0;i<=10;i++)
{
    if (i==3) continue;
    x=x+"The number is "+i+"<br>";
}
```

4.3 错误调试

4.3.1 try-catch-throw

try 语句测试代码块的错误。catch 语句处理错误。throw 语句创建自定义错误。当 JavaScript 引擎执行 JavaScript 代码时，会发生各种错误，可能是语法错误、来自服务器或用户错误输出而导致的错误、其他不可预知的错误。当发生错误产生问题时，JavaScript 引擎通常会停止，并生成一个错误消息。

1. try 和 catch

try 语句允许在程序中定义错误测试的代码块。catch 语句定义 try 代码块发生错误时所执行的代码块。try 和 catch 是成对出现的，如下所示。

```
try {
    //在这里运行代码
}
catch(err) {
    //在这里处理错误
}
```

在下面的例子中，try 块的代码中写了一个错字，catch 块会捕捉到 try 块中的错误，并执行错误处理程序来处理它。

```
var txt="";
function message()
{
    try { adddlert("Welcome guest!"); }
    catch(err) {
        txt="本页有一个错误。\n\n";
        txt+="错误描述："+err.message+"\n\n";
        txt+="单击确定继续。\n\n";
        alert(txt);
    }
}
```

2. throw 语句

throw 语句创建自定义错误。如果把 throw 与 try 和 catch 一起使用，则能够控制程序流，并生成自定义的错误消息。语法如下：

```
throw exception
```

异常可以是 JavaScript 字符串、数字、逻辑值或对象。

下面的例子检测输入变量的值。如果值是错误的,会抛出一个异常(错误)。catch 会捕捉到这个错误,并显示一段自定义的错误消息。

```
function myFunction()
{
    try
    {
        var x=document.getElementById("demo").value;
        if(x=="")    throw "值为空";
        if(isNaN(x)) throw "不是数字";
        if(x>10) throw "太大";
        if(x<5) throw "太小";
    }
    catch(err)
    {
        var y=document.getElementById("mess");
        y.innerHTML="错误: "+err+"。";
    }
}
```

如果 getElementById 函数出错,上面的例子也会抛出一个错误。

4.3.2 调试

程序可能包含语法错误、逻辑错误。如果没有调试工具,就难以发现这些错误。通常,如果 JavaScript 出现错误,又没有提示信息,就无法找到程序的错误位置。而在编写程序时,难免不发生错误。在程序代码中寻找错误叫做代码调试,很多浏览器都内置了调试工具。内置的调试工具可以开始或关闭,严重的错误信息会发送给用户。

有了调试工具,就可以设置断点 (代码停止执行的位置),且可以在代码执行时检测变量。浏览器启用调试工具一般是按下 F12 键,并在调试菜单中选择 Console 输出结果,或利用设置断点开辟观察变量窗口,或利用 try-catch 捕获异常。

1. console.log() 方法

如果浏览器支持调试,可以使用 console.log()方法在调试窗口上打印 JavaScript 值,如下所示。

```
a=5;
b=6;
c=a+b;
console.log(c);
```

2. 设置断点

在调试窗口中可以设置 JavaScript 代码的断点。在每个断点上都会停止执行

JavaScript 代码,以检查 JavaScript 变量的值。检查完毕可以重新执行代码。

3. 主要浏览器的调试工具

通常,在浏览器中启用调试工具是按下 F12 键,并在调试菜单中选择 Console。不同浏览器的调试工具基本相同,但启动方式不尽相同,具体如下。

(1) Chrome 浏览器

单击最右侧的"自定义与控制 Google Chrome"图标,在菜单中选择工具,在工具中选择开发者工具,最后选择 Console。

(2) Firefox 浏览器

打开浏览器,访问页面 http://www.getfirebug.com,按照说明安装 Firebug,而后按 F12 键进入开发者工具,最后选择 Console。

(3) Internet Explorer 浏览器

打开浏览器,在菜单中选择开发者工具,或按 F12 键进入开发者工具,最后选择 Console。

(4) Opera 浏览器

打开浏览器,在菜单中选择开发者,在开发者中选择开发者工具,最后选择 Console。

(5) Safari 浏览器

打开浏览器,右击,选择检查元素,在底部弹出的窗口中选择"控制台"。

4.4 上机实验样例

1. 实验目标

掌握浏览器脚本程序控制台输出和设置断点进行跟踪调试技术。

2. 实验任务

建立一个 HTML 文档,利用 JavaScript 计算 1～100 的整数累加和,利用控制台输出结果。

3. 实验步骤

(1) 源程序:4-3-1.htm

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
        <script type="text/javascript">
            function ComputeSum() {
                var i;
                var sum;
```

```
                for (i=0; i<100; i++) sum+=i;
                console.log(sum);
                return sum;
            }
        </script>
</head>
    <body>
        <script type="text/javascript">
            ComputeSum();
        </script>
    </body>
</html>
```

(2) 在浏览器中浏览

在浏览器中浏览的结果如图 4.2 所示。

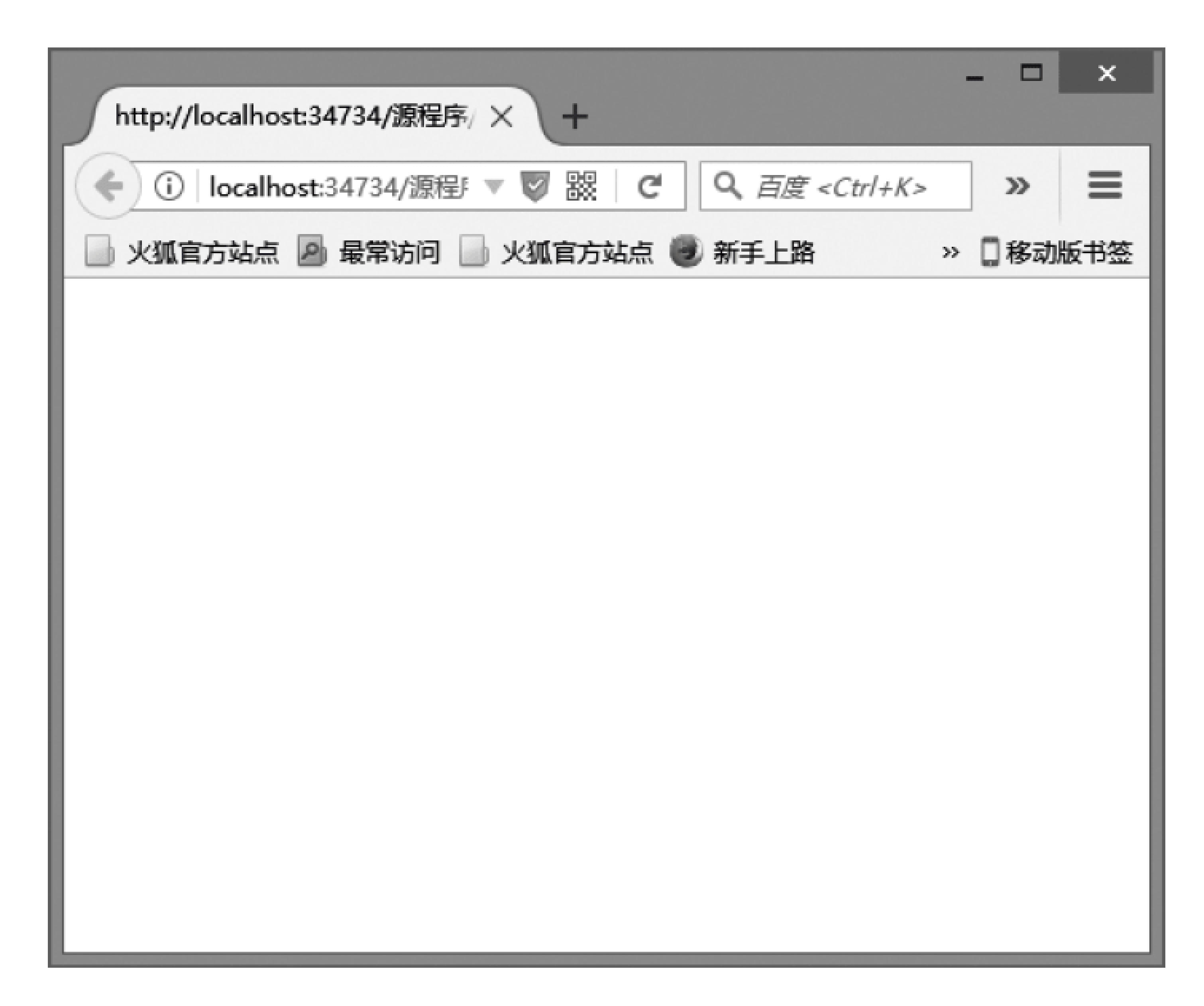


图 4.2 源程序 4-3-1 的浏览器浏览

(3) 在浏览器中的控制台中浏览

在图 4.2 所示的浏览器中按 F12 键,浏览器底部弹出开发人员工具界面,选中控制台选项卡,如图 4.3 所示。

图 4.3 所示的控制台中的输出为 NaN,需要单拍跟踪调试。步骤如下:

① 选中调试器:在开发人员工具中选择调试器。

② 选中 4-3-1.htm:在左侧的资源菜单中选中 4-3-1.htm。

③ 设置断点:在右侧的源程序中单击第 10 行的数字。

经过以上 3 步,断点设置成功,如图 4.4 所示。

(4) 设置监视 sum 变量

设置监视步骤如下。

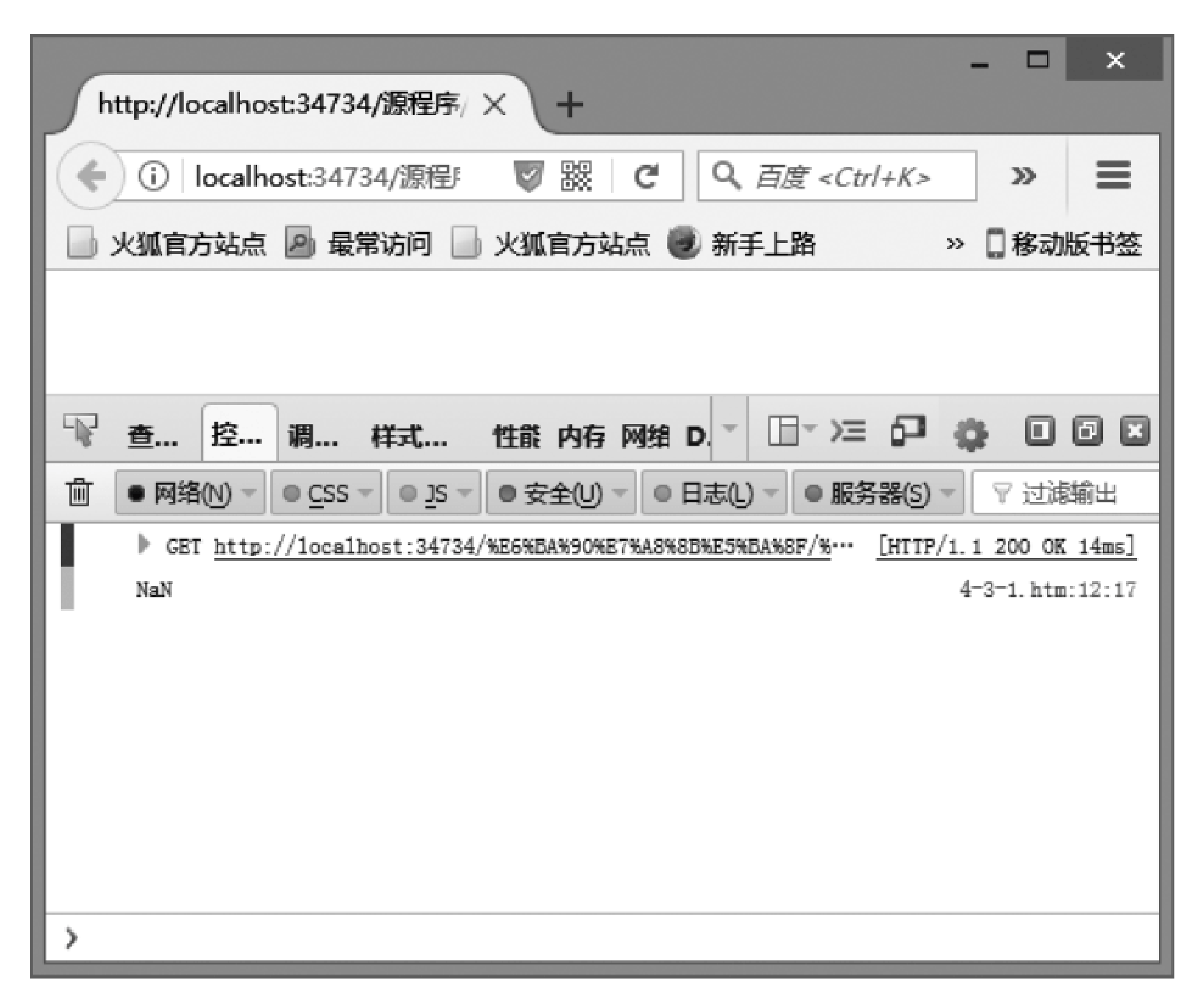


图 4.3　源程序 4-3-1 的浏览器浏览的控制台输出



图 4.4　成功设置断点后的界面

① 弹出快捷菜单。

在图 4.4 的右侧源程序区域右击，弹出如图 4.5 所示的快捷菜单。

② 输入监视变量 sum。

单击图 4.5 所示的“选择要监视的表达式”，在添加表达式文本框中输入 sum 后按回车键确认，显示如图 4.6 所示的界面。



图 4.5　调试器源程序区弹出的快捷菜单



图 4.6　添加监视 sum 变量

③ 在断点处观察 sum。

单击图 4.7 中所示的红色矩形圈中的按钮（“重新载入当前页面”按钮），则程序运行到断点处停止，如图 4.8 所示。观察右侧监视表达式，此时 sum 为 undefined。这意味着 sum 变量没有初值。按 F10 键 1 次，单步运行 1 次，执行 1 条语句，此时 sum 变成 NaN，如图 4.9 所示。



图 4.7　单击重新加载页面按钮

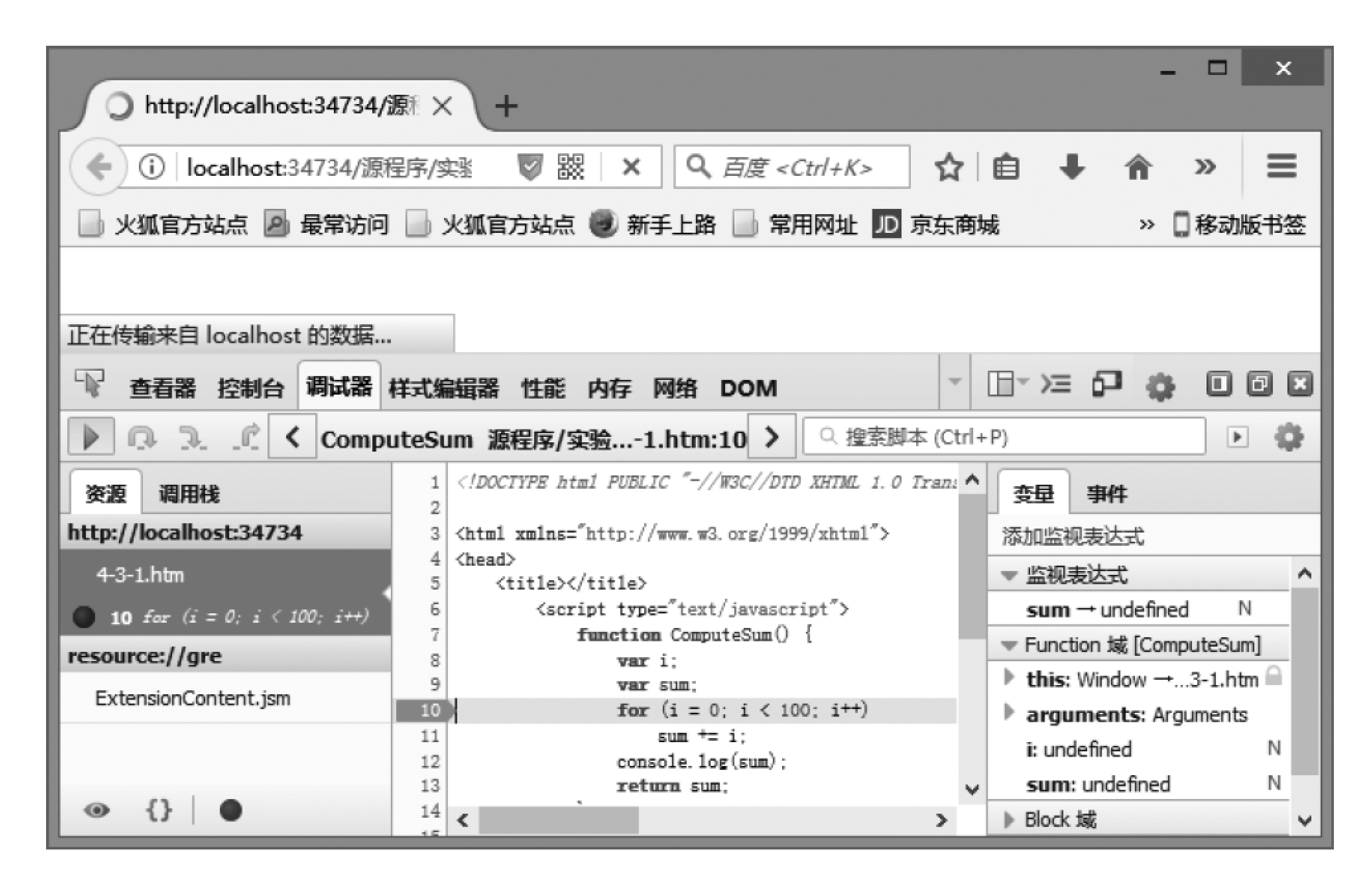


图 4.8　运行到断点处的 sum 值

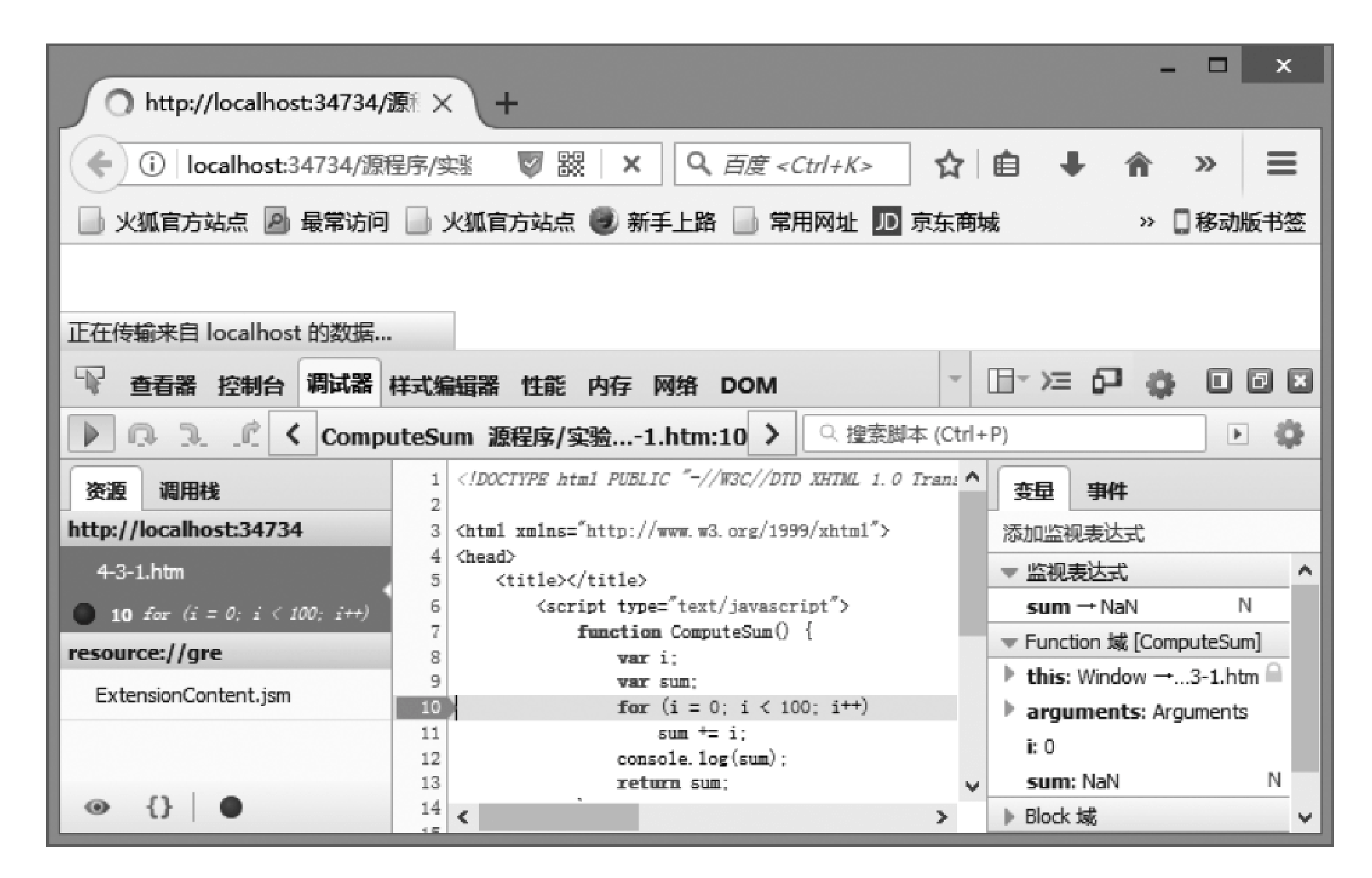


图 4.9 单步运行 1 句的 sum 值

(5) 修正源程序

将 var sum;改成 var sum=0。

(6) 取消断点

在调试器右侧的源程序中单击语句 10 的标号 10,取消断点。

(7) 重新加载页面

单击图 4.7 中红色矩形圈中的按钮,重新加载页面,如图 4.10 所示,此时控制台输出 4950。



图 4.10 为 sum 赋初值取消断点重新加载页面后控制台的输出

4.5 实验任务

1. 实验题目

实现杨辉三角形 HTML 浏览器开发工具控制台输出程序。

2. 实验目的

JavaScript 程序错误跟踪调试与异常捕获技能训练。

3. 实验类型

验证与技能训练。

4. 遇到的问题及解决办法

(1) 给出至少一处错误的描述。
(2) 给出修正错误前设定的断点位置。
(3) 给出修正错误所监视的变量。
(4) 给出运行到断点处监视变量值。
(5) 给出错误修正后控制台运行结果。
(6) 给出至少一个异常语句的捕获。
(7) 以上全部操作用屏幕截图支撑。

5. 实验环境

(1) 计算机：PC、内存 8GB、主频 1.8GHz 及以上、硬盘 500GB 及以上。
(2) 操作系统：Windows XP、Windows 7、Windows 8、Windows 10。
(3) 开发环境：Visual Studio 2010 或 Adobe Dreamweaver。
(4) 浏览器：IE9 及以上、Chrome、Firefox、Safari、Edge、QQ 浏览器等。

6. 实验原理

(1) 杨辉三角形的二维坐标。
(2) 二维坐标与控制台的映射。
(3) 二重循环对应二维坐标。

7. 源程序

(1) 辅以必要的注释。
(2) 错落有致，结构清晰，易读性强。

第 5 章



HTML DOM

■ **知识目标**

- 理解 DOM 的概念
- 掌握 DOM CSS 的应用
- 掌握 DOM 事件的响应与处理
- 掌握 DOM 添加删除元素功能的实现
- 掌握 BOM 的 Windows 对象方法
- 掌握 BOM 的计时事件
- 掌握 BOM 的 Cookie 概念和应用

■ **能力目标**

- 根据实际需求恰当选择事件并进行相应处理
- 根据具体要求动态改变 HTML 元素样式
- 根据功能需要设计与实现动态 HTML 交互页面
- 根据实际需求恰当应用计时事件和 Cookie 技术

■ **素质目标**

- 根据实际需要设计与实现动态交互 HTML 页面
- 根据实际应用需求编写恰当的键盘与鼠标事件处理程序

■ **教学重点**

- DOM 事件响应与处理
- DOM 改变元素样式
- BOM 计时事件应用

■ **教学难点**

- 对象的可见性、生存期与重叠

■ **建议学时**

- 理论：2 学时
- 实验：2 学时

5.1 概　　述

文档对象模型，即 DOM(Document Object Model)，独立于平台和语言，以统一方式访问和修改一个文档的内容和结构。DOM 以对象管理组织(OMG)的规约为基础，可用于任何编程语言。DOM 技术使得用户页面可以动态地变化，它可以显示或隐藏一个元素，改变元素的属性，增加一个元素等，这使得页面的交互性大大增强。

DOM 是以面向对象方式描述的文档模型，定义了表示和修改文档所需的对象、这些对象的行为和属性以及这些对象之间的关系。DOM 把页面元素组织成一个树形结构(可能并不是以这种树的方式具体实现)。

通过 JavaScript 可以重构整个 HTML 文档，可以添加、移除、改变或重排页面上的元素。JavaScript 可以获得对 HTML 文档中所有元素访问的入口。根据这个入口，应用对 HTML 元素进行添加、移动、改变或移除的方法和属性，可以组织动态 HTML 页面，实现用户与页面的动态交互。

1998 年，W3C 发布了第一级 DOM 规范。这个规范允许访问和操作 HTML 页面中每一个单独的元素。所有浏览器都已执行该标准，从而解决 DOM 的兼容性问题。DOM 可被 JavaScript 用来读取、改变 HTML、XHTML 及 XML 文档。

1. 基本构成

DOM 遵循 W3C(万维网联盟)的标准，定义了访问 HTML 和 XML 文档的标准。W3C 文档对象模型是中立于平台和语言的接口，允许程序和脚本动态地访问和更新文档的内容、结构和样式，分为核心、XML 及 HTML 三大部分。

(1) 核心 DOM

核心是针对任何结构化文档的标准模型。

(2) XML DOM

XML DOM 是针对 XML 文档的标准模型，提供访问和操纵 XML 的标准编程接口，中立于平台和语言，遵从 W3C 标准。XML DOM 定义了所有 XML 元素的对象和属性，以及访问它们的方法(接口)，即 XML DOM 是用于获取、更改、添加或删除 XML 元素的标准。

(3) HTML DOM

HTML DOM 是针对 HTML 文档的标准模型，提供访问和操纵 HTML 的标准编程接口，遵从 W3C 标准，定义了所有 HTML 元素的对象和属性，以及访问它们的方法。换言之，HTML DOM 是关于获取、修改、添加或删除 HTML 元素的标准。

2. 文档树

HTML 文档中的所有节点组成了一个文档树(或节点树)。节点彼此间都有等级关系。HTML 文档中的每个元素、属性、文本等都代表着树中的一个节点。树起始于文档节点，并由此继续伸出枝条，直到处于这棵树最低级别的所有文本节点为止。文档树示

例如图 5.1 所示。

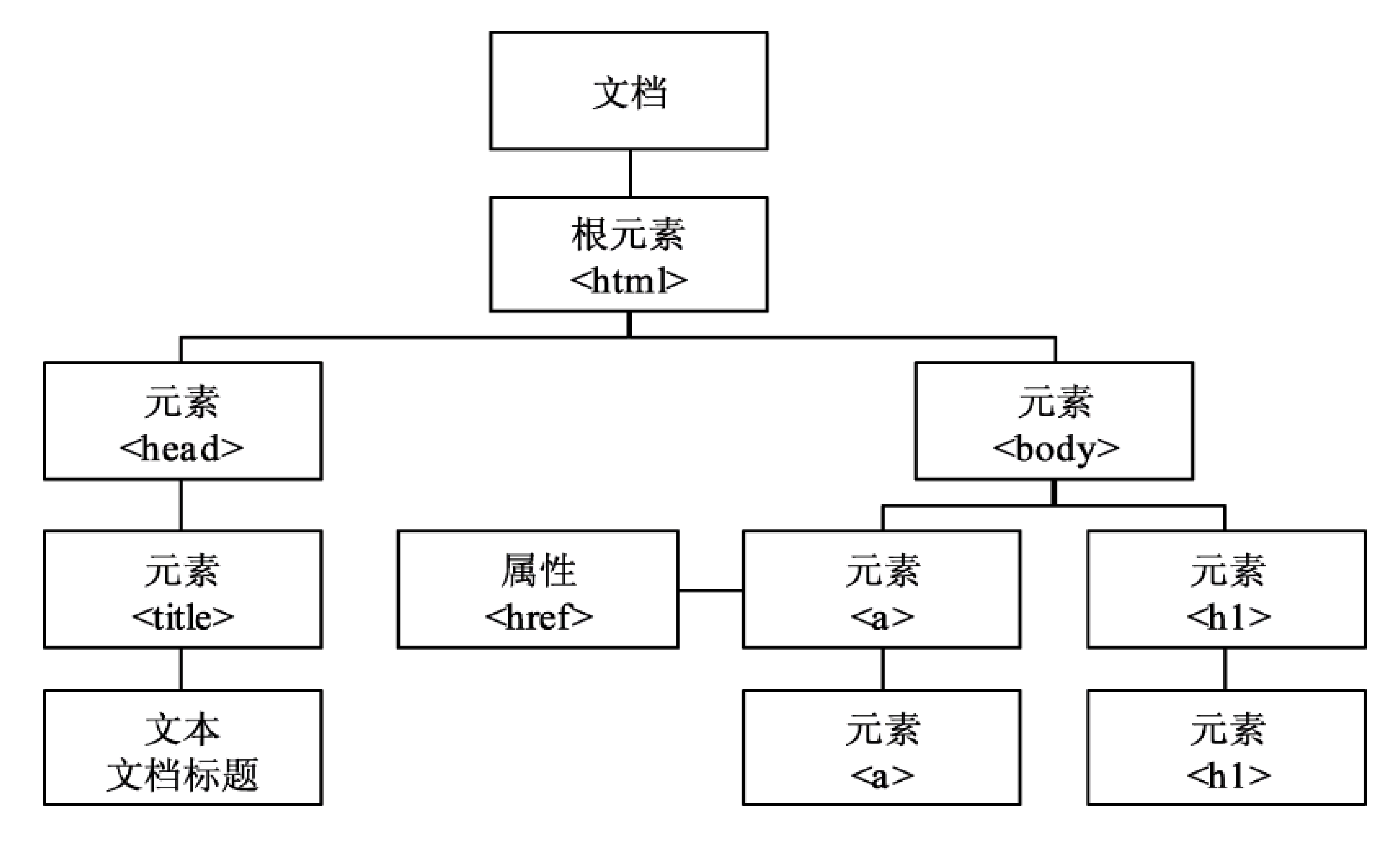


图 5.1　文档树示例

图 5.1 中的所有节点彼此间都存在关系。除根元素节点之外的每个节点都有父节点，如<head>和<body> 的父节点是 <html> 节点，元素 a 节点的父节点是 <body> 节点。

大部分元素节点都有子节点，如<head>节点有一个子节点<title>节点。<title>节点有一个文本子节点。

当节点分享同一个父节点时，它们就是同辈(同级节点)，如<h1>和<a>是同辈，因为它们的父节点均是<body>节点。

节点也可以拥有后代，后代指某个节点的所有子节点，或者这些子节点的子节点，以此类推。例如，所有文本节点都是 <html>节点的后代，而第一个文本节点是<head>节点的后代。

节点也可以拥有先辈。先辈是某个节点的父节点，或者父节点的父节点，以此类推。例如，所有的文本节点都可把 <html> 节点作为先辈节点。

3. 访问结点

可使用如下方法来查找所操作元素。

- getElementById()和 getElementsByTagName()方法。
- 节点的 parentNode、firstChild 以及 lastChild 属性。

(1) getElementById()和 getElementsByTagName()方法

这两种方法可查找整个 HTML 文档中的任何 HTML 元素。它们会忽略文档的结构。如果希望查找文档中所有的<p>元素，getElementsByTagName()会找到全部，不管<p>元素处于文档中的哪个层次。同时，getElementById()方法也会返回正确的元素，不论它被隐藏在文档结构中的什么位置。这两种方法会找到任何 HTML 元素，不论它们在文档中所处的位置。

getElementById()可通过指定的 ID 返回元素，语法如下：

```
document.getElementById("ID");
```

getElementById()无法工作在 XML 中。在 XML 文档中，通过拥有类型 id 的属性进行搜索，而此类型必须在 XML DTD 中声明。

例如，document. getElementById('maindiv'). getElementsByTagName("p")；会返回所有<p>元素的一个节点列表，且这些<p>元素必须是 id 为"maindiv"的元素的后代。

getElementsByTagName()方法会使用指定的标签名返回所有元素(作为一个节点列表)，这些元素是使用此方法时所处的元素的后代。

getElementsByTagName()可被用于任何的 HTML 元素，语法如下：

```
document.getElementsByTagName("标签名称");
```

例如，document. getElementsByTagName("p")；会返回文档中所有<p>元素的一个节点列表。

使用节点列表时，通常要把此列表保存在一个变量中，例如：

```
var x=document.getElementsByTagName("p");
```

变量 x 是包含着页面中所有<p>元素的一个列表，并且可以通过它们的索引号(从 0 开始)来访问这些<p>元素。

可以通过使用 length 属性来循环遍历节点列表，例如：

```
var x=document.getElementsByTagName("p");
for (var i=0;i<x.length;i++) { //do something with each paragraph }
```

也可以通过索引号来访问某个具体的元素。比如访问第三个<p>元素，写成：

```
var y=x[2];
```

(2) parentNode、firstChild 及 lastChild 属性

这 3 个属性可遵循文档的结构，在文档中访问结点。

下面展示这个 HTML 片段：

```
<table>
    <tr>
        <td>John</td>
        <td>Doe</td>
        <td>Alaska</td>
    </tr>
</table>
```

在上面的 HTML 代码中，第一个<td>是<tr>元素的首个子元素(firstChild)，最后一个<td>是<tr>元素的最后一个子元素(lastChild)。此外，<tr>是每个<td>元素的父节点(parentNode)。

5.2 DOM 功能

通过可编程的对象模型,JavaScript 能创建动态的 HTML,具体功能如下:

① 改变页面中的 HTML 元素。

② 改变页面中的 HTML 属性。

③ 改变页面中的 CSS 样式。

④ 对页面中的事件做出反应。

5.2.1 DOM HTML

HTML DOM 允许 JavaScript 改变 HTML 元素的内容。

1. 改变 HTML 输出流

JavaScript 能够创建动态的 HTML 内容,在 JavaScript 中,document.write()可用于直接向 HTML 输出流中写内容。

例 5-2-1-1 浏览器输出为:“今天的时间是:Thu Feb 16 2017 09:03:00 GMT+0800”,源文档如下:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
    </head>
    <body>
        <script>
            document.write(Date());
        </script>
    </body>
</html>
```

注意:文档加载完成之后使用 document.write(),会覆盖该文档。

2. 改变 HTML 内容

修改 HTML 内容最简单的方法是使用 innerHTML 属性。改变 HTML 元素内容的语法如下:

```
document.getElementById(id).innerHTML=新的 HTML
```

例 5-2-1-2 改变<p>元素的内容,源文档如下:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
    <head>
        <title></title>
    </head>
    <body>
        <p id="p1">Hello World!</p>
        <script>
            document.getElementById("p1").innerHTML="新文本!";
        </script>
    </body>
</html>
```

3. 改变 HTML 属性

改变 HTML 元素的属性语法如下：

```
document.getElementById(id).attribute=新属性值
```

例 5-2-1-3 改变图片。

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
    </head>
    <body>
        <img id="image" src="../Images/0.jpg" alt=""/>
        <script>
            document.getElementById("image").src="../Images/1.jpg";
        </script>
    </body>
</html>
```

例 5-2-1-3 的 HTML 文档含有 id="image"的<img>元素，用 HTML DOM 获得 id="image" 的元素，然后更改此元素的属性。

5.2.2 DOM CSS

HTML DOM 允许 JavaScript 改变 HTML 元素的样式。改变 HTML 元素样式的语法如下：

```
document.getElementById(id).style.property=新样式
```

例 5-2-2-1 改变 <p> 元素的样式，源文档如下：

```
<!DOCTYPE html>
<html>
    <head>
```

```
        <title>以上段落通过脚本修改</title>
    </head>
    <body>
        <p id="p1">Hello World!</p>
        <p id="p2">Hello World!</p>
        <script>
            document.getElementById("p2").style.color="blue";
            document.getElementById("p2").style.fontFamily="Arial";
            document.getElementById("p2").style.fontSize="30px";
        </script>
    </body>
</html>
```

5.2.3 DOM事件

HTML DOM 能通过触发事件来执行代码。HTML 有许多不同类型的事件，包括单击、网页加载、图像加载、鼠标指向元素、输入文本改变、提交表单、触发按键等。

1. 事件分类

（1）一般事件

onClick：单击事件，多用在某个对象控制范围内单击。

onDblClick：双击事件。

onMouseDown：鼠标上按钮被按下。

onMouseUp：鼠标按下后，松开时激发的事件。

onMouseOver：鼠标移动到某对象范围上方时触发的事件。

onMouseMove：鼠标移动时触发的事件。

onMouseOut：鼠标离开某对象范围时触发的事件。

onKeyPress：键盘上的某个键被按下并且释放时触发的事件。

onKeyDown：键盘上某个按键被按下时触发的事件。

onKeyUp：键盘上某个按键被按后放开时触发的事件。

（2）页面相关事件

onAbort：图片下载时被用户中断。

onBeforeUnload：当前页面的内容将要被改变时触发的事件。

onError：捕捉当前页面出现的错误，如脚本错误与外部数据引用的错误。

onLoad：页面加载完成传送到浏览器时触发的事件，包括外部文件引入完成。

onMove：浏览器的窗口被移动时触发的事件。

onResize：浏览器的窗口大小被改变时触发的事件。

onScroll：浏览器的滚动条位置发生变化时触发的事件。

onStop：浏览器的停止按钮被按下时触发的事件或者正在下载的文件被中断。

onUnload：当前页面将被改变时触发的事件。

(3) 表单相关事件

onBlur：当前元素失去焦点时触发的事件，鼠标与键盘的触发均可。

onChange：当前元素失去焦点且元素内容发生改变而触发的事件，鼠标与键盘的触发。

onFocus：某个元素获得焦点时触发的事件。

onReset：表单中 RESET 的属性被激发时触发的事件。

onSubmit：一个表单被递交时触发的事件。

(4) 滚动字幕事件

onBounce：在 Marquee 内的内容移动至 Marquee 显示范围之外时触发的事件。

onFinish：Marquee 元素完成需要显示的内容后触发的事件。

onStart：Marquee 元素开始显示内容时触发的事件。

(5) 编辑事件

onBeforeCopy：当页面被选内容将要复制到浏览者系统的剪贴板前触发的事件。

onBeforeCut：当页面一部分或者全部内容将被移离当前页面剪贴并移动到浏览者的系统剪贴板时触发的事件。

onBeforeEditFocus：当前元素将要进入编辑状态。

onBeforePaste：内容将要从浏览者的系统剪贴板传送粘贴到页面中时触发的事件。

onBeforeUpdate：当浏览者粘贴系统剪贴板中的内容时通知目标对象。

onContextMenu：当浏览者右击出现菜单时，或者通过键盘的按键触发页面菜单时触发的事件。

onContentMenu="return false"禁止使用鼠标右键。

onCopy：当页面被选内容被复制后触发的事件。

onCut：当页面被选内容被剪切时触发的事件。

onDrag：当某个对象被拖动时触发的事件。

onDragDrop：一个外部对象被鼠标拖进当前窗口或者帧。

onDragEnd：当鼠标拖动结束时触发的事件，即释放鼠标按钮。

onDragEnter：当对象被鼠标拖动的对象进入其容器范围内时触发的事件。

onDragLeave：当对象被鼠标拖动的对象离开其容器范围内时触发的事件。

onDragOver：当某被拖动的对象在另一对象容器范围内拖动时触发的事件。

onDragStart：当某对象将被拖动时触发的事件。

onDrop：在一个拖动过程中释放鼠标键时触发的事件。

onLoseCapture：当元素失去鼠标移动所形成的选择焦点时触发的事件。

onPaste：当内容被粘贴时触发的事件。

onSelect：当文本内容被选择时的事件。

onSelectStart：当文本内容选择将开始发生时触发的事件。

(6) 数据绑定

onAfterUpdate：当数据完成由数据源到对象的传送时触发的事件。

onCellChange：当数据来源发生变化时。

onDataAvailable：当数据接收完成时触发事件。

onDatasetChanged：数据在数据源发生变化时触发的事件。

onDatasetComplete：当来自数据源的全部有效数据读取完毕触发的事件。

onErrorUpdate：当使用 onBeforeUpdate 事件触发取消了数据传送时，代替 onAfterUpdate 事件。

onRowEnter：当前数据源的数据发生变化，并且有新的有效数据时触发的事件。

onRowExit：当前数据源的数据将要发生变化时触发的事件。

onRowsDelete：当前数据记录将被删除时触发的事件。

onRowsInserted：当前数据源将要插入新数据记录时触发的事件。

（7）外部事件

onAfterPrint：当文档被打印后触发的事件。

onBeforePrint：当文档即将打印时触发的事件。

onFilterChange：当某个对象的滤镜效果发生变化时触发的事件。

onHelp：当浏览者按下 F1 键或者浏览器的帮助选择时触发的事件。

onPropertyChange：当对象的属性之一发生变化时触发的事件。

onReadyStateChange：当对象的初始化属性值发生变化时触发的事件。

2. 事件触发

向 HTML 元素分配事件，使用事件属性，设置事件属性具体动作和行为。

例 5-2-3-1 当在<h1>元素上单击时，改变其内容，源文档如下：

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
    </head>
    <body>
        <h1 onclick="this.innerHTML='Ooops!'">单击文本!</h1>
    </body>
</html>
```

例 5-2-3-2 当用户在 <h1> 元素上单击时，事件处理器调用一个函数改变其内容，源文档如下：

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
        <script>function changetext(id) { id.innerHTML="Ooops!"; }</script>
    </head>
    <body>
        <h1 onclick="changetext(this)">单击文本!</h1>
```

```
    </body>
</html>
```

HTML DOM 允许使用 JavaScript 向 HTML 元素分配事件，如例 5-2-3-3 所示。

例 5-2-3-3 单击按钮一次，刷新一次当前时间，源文档如下：

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
        <script>
            function displayDate() { document.getElementById("myDate").
            value=Date(); };
        </script>
    </head>
    <body>
          <label>当前时间:</label>
        <input type="text" id="myDate" style="width:150px;"/>
        <input type="button" id="myBtn" value="刷新日期"/>
        <script>
            displayDate();
            document.getElementById("myBtn").onclick=function ()
            { displayDate() };
        </script>
    </body>
</html>
```

在例 5-2-3-3 中，名为 displayDate 的函数被分配给 id="myBtn"的 HTML 元素。单击按钮时 displayDate 函数将会被执行。

5.2.4 HTML DOM元素

1. 添加 HTML 元素

向 HTML DOM 添加新元素，首先创建该元素（元素节点），然后向一个已存在的元素追加该元素，如例 5-2-4-1 所示。

例 5-2-4-1 向一个 div 里添加元素 p，元素 p 内含文本节点“这是一个新段落”。

（1）源文档

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
    </head>
    <body>
```

```
        <div id="div1">
            <p id="p1">这是一个段落。</p>
            <p id="p2">这是另一个段落。</p>
        </div>
        <script>
            var para=document.createElement("p");
            var node=document.createTextNode("这是一个新段落。");
            para.appendChild(node);
            var element=document.getElementById("div1");
            element.appendChild(para);
        </script>
    </body>
</html>
```

(2) 运行结果

浏览器显示结果如图 5.2 所示。

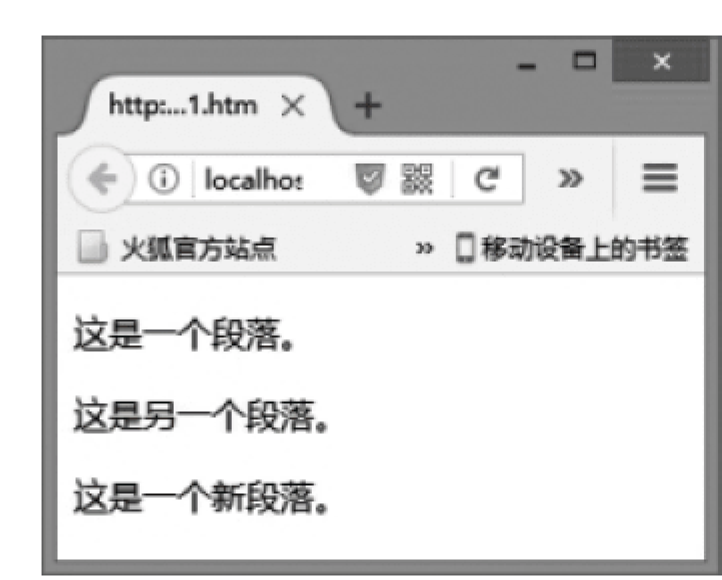


图 5.2　例 5-2-4-1.htm 的浏览器显示结果

(3) 实例解析

var para＝document.createElement("p")；语句用于创建新的<p>元素。

var node＝document.createTextNode("这是一个新段落。")；语句用于创建文本节点。

para.appendChild(node)；语句向<p>元素追加文本节点。

var element＝document.getElementById("div1")；语句找到一个已有 div1 元素。

element.appendChild(para)；向一个 div1 元素追加这个新元素。

2. 删除 HTML 元素

例 5-2-4-2 演示了删除元素操作过程。

例 5-2-4-2　从 div1 中删除 p2 元素。

(1) 源文档

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
    </head>
    <body>
        <div id="div1">
            <p id="p1">这是一个段落。</p>
            <p id="p2">这是另一个段落。</p>
        </div>
        <script>
            var parent=document.getElementById("div1");
            var child=document.getElementById("p1");
```

```
            parent.removeChild(child);
        </script>
    </body>
</html>
```

(2) 运行结果

浏览器显示如图 5.3 所示。



图 5.3　例 5-2-4-2.htm 的浏览器显示

(3) 实例解析

var parent=document.getElementById("div1");语句找到 id="div1"的元素。

var child=document.getElementById("p1");语句找到 id="p1"的<p>元素。

parent.removeChild(child);语句从父元素中删除子元素。

5.3　HTML BOM

BOM(Browser Object Document)是浏览器对象模型,提供独立于内容而与浏览器窗口进行交互的功能。BOM 主要用于管理窗口与窗口之间的通信,其核心对象是 window。BOM 由一系列相关的对象构成,并且每个对象都提供了很多方法与属性。

BOM 包含一些对象,提供不同功能。window 对象可以移动、调整浏览器大小,location 对象和 history 对象可以导航,navigator 与 screen 对象可以获取浏览器、操作系统与用户屏幕信息,document 对象可以使用作为访问 HTML 文档的入口,frames 对象用于管理框架。BOM 对象隶属关系如图 5.4 所示。

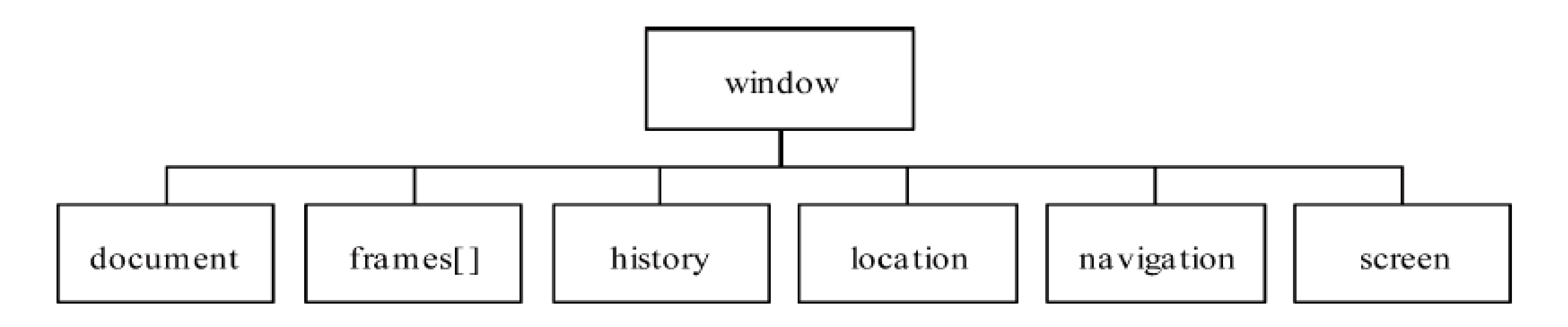


图 5.4　BOM 对象隶属关系

5.3.1　window 对象

BOM 使 JavaScript 有能力与浏览器"对话"。由于现代浏览器已经实现了 JavaScript 交互性方面的相同方法和属性,常被认为是 BOM 的方法和属性。

所有浏览器都支持 window 对象,它表示浏览器窗口。所有 JavaScript 全局对象、函数以及变量均自动成为 window 对象的成员。全局变量是 window 对象的属性。全局函数是 window 对象的方法。HTML DOM 的 document 也是 window 对象的属性之一,如下所示:

```
window.document.getElementById("header");
```

为简化表达,实际中常省略 window,写成:

```
document.getElementById("header");
```

1. Window 尺寸

有 3 种方法能够确定浏览器窗口的尺寸(浏览器的窗口,不包括工具栏和滚动条)。

(1) Internet Explorer、Chrome、Firefox、Opera 及 Safari 浏览器的确定方法

高度:window.innerHeight——浏览器窗口的内部高度;

宽度:window.innerWidth——浏览器窗口的内部宽度。

(2) Internet Explorer 8、7、6、5 浏览器的确定方法

高度:document.documentElement.clientHeight;

宽度:document.documentElement.clientWidth;

或者

高度:document.body.clientHeight;

宽度:document.body.clientWidth;

(3) 实用的 JavaScript 方案(适用于所有浏览器)

宽度:var h=window.innerHeight || document.documentElement.clientHeight
|| document.body.clientHeight;

高度:var w=window.innerWidth || document.documentElement.clientWidth
|| document.body.clientWidth;

2. 一些常用的方法

一些常用的方法如表 5.1 所示。演示如例 5-3-1-1 所示。

表 5.1 一些常用的方法

方　法	描　述
window.alert("提示信息")	显示包含消息的警示框
window.confirm("确认信息")	显示一个确认对话框,包含确定和取消按钮
window.prompt("提示信息")	弹出提示信息框
window.open("url","name")	打开具有指定名称的新窗口,并加载指定 URL 给定的文档;如果没有提供 URL,则打开一个空文档
window.close()	关闭顶层浏览器窗口。某个窗口可以通过调用 self.close()或只调用 close()来关闭其自身。只有通过 JavaScript 代码打开的窗口才能由 JavaScript 代码关闭。这阻止了恶意的脚本终止用户的浏览器
window.moveTo(x,y)	移动当前窗口到(x,y)处
window.resizeTo(width,height)	调整当前窗口宽为 width,高为 height,单位是像素

例 5-3-1-1 打开文档示例。

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
    <head>
        <title></title>
        <script type="text/javascript">
            function openWindow() {
                myWindows=window.open("http://www.ldu.edu.cn", "鲁东大学");
            }
            function closeWindow() { myWindows.close(); }
        </script>
    </head>
    <body>
        <input type='button' value='鲁东大学' onclick='openWindow();' />
        <input type='button' value='关闭鲁东大学' onclick='closeWindow();' />
        <input type='button' value='alert演示' onclick='alert("出现错误!");' />
        <input type='button' value='prompt演示' onclick='prompt("成功关闭");' />
        <input type='button' value='confirm演示' onclick='confirm("结束演示?");' />
    </body>
</html>
```

5.3.2 计时事件

计时事件是 BOM 的一个重要方法。在一个设定时间间隔之后执行 JavaScript 代码，而不是在函数被调用后立即执行，被称为计时事件。

在 JavaScript 中使用计时事件有如下两个方法：

① setInterval()：间隔指定的毫秒数不停地执行指定的代码。

② setTimeout()：暂停指定的毫秒数后执行指定的代码。

1. setInterval() 方法

setInterval() 间隔指定的毫秒数不停地执行指定的代码，语法如下：

```
window.setInterval("javascript function",milliseconds);
```

window. setInterval()方法可以不使用 window 前缀，直接使用函数 setInterval()。

setInterval()的第一个参数是函数(function)，第二个参数是间隔的毫秒数(1000 毫秒是 1 秒)。

例如：每 3 秒弹出 "hello"，可写成

```
setInterval(function(){alert("Hello")},3000);
```

例 5-3-2-1 类似手表一样显示当前时间，源文档如下。

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
```

```
    </head>
    <body>
        <label id="m_date"></label>
        <script type="text/javascript">
            var myVar=setInterval(function () { myTimer() }, 1000);
            function myTimer() {
                var d=new Date();
                var t=d.toLocaleTimeString();
                document.getElementById("m_date").innerHTML=t;
            }
        </script>
    </body>
</html>
```

2. clearInterval()方法

clearInterval()方法用于停止 setInterval()方法执行的函数代码，语法如下：

```
window.clearInterval(intervalVariable);
```

window. clearInterval()方法可以不使用 window 前缀，直接使用函数 clearInterval()。使用 clearInterval()方法前，创建计时方法时使用全局变量，语句如下：

```
myVar=setInterval("javascript function",milliseconds);
```

然后使用 clearInterval()方法停止执行。

例 5-3-2-2 类似手表一样显示当前时间，按停止按钮停止计时，按启动按钮继续计时，源文档如下。

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
    </head>
    <body>
        <input type="button" onclick="Start();" value="启动"/>
        <input type="button" onclick="Stop();" value="停止"/>
        <label id="m_date"></label>
        <script type="text/javascript">
            var myTimer;
            function Start() { if(myTimer==null) myTimer=setInterval(Run,
            1000);}
            function Run() {
                var d=new Date();
                var t=d.toLocaleTimeString();
                document.getElementById("m_date").innerHTML=t;
```

```
            }
            function Stop() { if (myTimer != null) { clearInterval(myTimer); myTimer=
           null; } }
            Start();
        </script>
    </body>
</html>
```

setTimeout()和 clearTimeout()方法与 setInterval()和 clearInterval()方法的用法类似,不再重复。

5.3.3 Cookie

Cookie 用于存储 Web 页面的用户信息,将用户数据以文本形式存储于本地电脑的文件中。当 Web 服务器向浏览器发送 Web 页面时,连接关闭后,服务器端不会记录用户的信息。Cookie 的作用是记录客户端的用户信息。用户访问 Web 页面时,用户名、密码等信息可以记录在 Cookie 中。下一次访问该页面时,可以在 Cookie 中读取用户访问记录。Cookie 以名/值对形式存储,如下所示:

```
username=John Doe
```

当浏览器从服务器上请求 Web 页面时,属于该页面的 cookie 会被添加到该请求中。服务器端通过这种方式获取用户的信息。JavaScript 可以使用 document.cookie 属性来创建、读取及删除 cookie。

1. 创建 Cookie

JavaScript 创建 cookie 如下所示:

```
document.cookie="username=John Doe";
```

可以为 cookie 添加一个过期时间(以 UTC 或 GMT 时间为准)。在默认情况下,cookie 在浏览器关闭时删除。例如,用户 John Doe 的过期时间为 2013 年 12 月 18 日 12 点整,写成:

```
document.cookie="username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 GMT";
```

可以使用 path 参数告诉浏览器 cookie 的路径。默认情况下,cookie 属于当前页面。下面语句中的 path=/表示设为根目录:

```
document.cookie="username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 GMT; path=/";
```

2. 读取 Cookie

JavaScript 读取 cookie 如下所示:

```
var x=document.cookie;
```

document.cookie 以字符串方式返回所有的 cookie,类型格式如下:

```
cookie1=value; cookie2=value; cookie3=value;
```

3. 修改 Cookie

JavaScript 修改 cookie 类似于创建 cookie,新的 cookie 覆盖旧的 cookie,如下所示:

```
document.cookie="username=John Smith; expires=Thu, 18 Dec 2013 12:00:00 GMT; path=/";
```

4. 删除 Cookie

删除 cookie 需要设置 expires 参数为以前的时间即可,不必指定 cookie 的值,如下所示,设置为 Thu, 01 Jan 1970 00:00:00 GMT:

```
document.cookie="username=; expires=Thu, 01 Jan 1970 00:00:00 GMT";
```

5. Cookie 字符串

document.cookie 属性看起来像一个普通的文本字符串,其实不是。即使在 document.cookie 中写入一个完整的 cookie 字符串,重新读取 cookie 信息时,cookie 信息也是以名/值对形式展示的。

如果设置了新 cookie 数据项,原有的 cookie 数据项不会被覆盖。新 cookie 数据项将添加到 document.cookie 中,所以如果重新读取 document.cookie,将获得如下数据:

```
cookie1=value; cookie2=value;
```

如果查找一个指定 cookie 值,须创建一个 JavaScript 函数,在 cookie 字符串中查找 cookie 值。

5.4 上机实验样例

1. 实验目的

掌握 cookie 的应用。

2. 功能描述

创建 cookie,存储访问者名称。访问者下一次访问页面时,会看到一个欢迎的消息。首先,访问者访问 Web 页面,填写自己的名字,该名字会存储在 cookie 中。创建如下 3 个函数,以封装 cookie 的访问。

(1) 设置 cookie 值的函数。
(2) 获取 cookie 值的函数。
(3) 检测 cookie 值的函数。

3. 实验步骤

（1）启动 VS2010。

（2）打开网站"HTML 实用网页设计技术/源程序/实验/5-3-3-cookie"。

（3）打开 cookie.js，源程序如下。

```
function setCookie(cname,cvalue,exdays)
{
    var d=new Date();
    d.setTime(d.getTime()+(exdays*24*60*60*1000));
    var expires="expires="+d.toGMTString();
    document.cookie=cname+"="+cvalue+"; "+expires;
}
function getCookie(cname)
{
    var name=cname+"=";
    var ca=document.cookie.split(';');
    for(var i=0; i<ca.length; i++)
    {
        var c=ca[i].trim();
        if (c.indexOf(name)==0) return c.substring(name.length,c.length);
    }
    return "";
}
function checkCookie()
{
    var username=getCookie("username");
    if (username!="")  {  alert("Welcome again "+username);  }
    else
    {
        username=prompt("Please enter your name:","");
        if (username!="" && username!=null)
        { setCookie("username",username,365); }
    }
}
```

setCookie(cname, cvalue, exdays)函数设置 cookie 值。其函数的参数中，cookie 的名称为 cname，cookie 的值为 cvalue，并设置了 cookie 的过期时间 expires。该函数设置了 cookie 名、cookie 值、cookie 过期时间。

getCookie(cname)函数获取 cookie 值。cookie 名的参数为 cname。

checkCookie()创建了一个检测 cookie 是否创建的函数。如果设置了 cookie，将显示一个问候信息。如果没有设置 cookie，将会显示一个弹窗，用于询问访问者的名字，并调用 setCookie 函数，将访问者的名字存储 365 天。

(4) 双击 cookie. htm 文档,源文档如下。

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
        <script src="cookie.js" type="text/javascript"></script>
    </head>
    <body onload="checkCookie();">
    </body>
</html>
```

(5) 在浏览器中首次浏览 cookie. htm,如图 5.5 所示。



图 5.5 首次浏览 cookie. htm 界面

在图 5.5 中输入 1234,单击"确定"按钮后,结果如图 5.6 所示。单击浏览器,重新载入当前页面图标,结果如图 5.7 所示。



图 5.6 单击图 5.5 的确定按钮后的界面



图 5.7 第二次浏览 cookie. htm 的界面

5.5 实验任务

1. 实验题目

DOM事件响应脚本程序设计。

2. 程序功能

综合利用所学的标签和CSS设计一个画图网页,含横向水平主菜单、纵向树形面板和客户区。横向水平主菜单含有长方形、圆、菱形、正方形、椭圆5个菜单项。纵向树形面板含有线型、颜色、动静3个选项,线型含有实线、虚线2个子项,颜色含有红、绿、蓝3个子项,动静含有动和静2个子选项。单击长方形,客户区显示长方形。单击正方形,客户区显示正方形。单击线型,当前图形线型改变。单击颜色,当前图形的边框和填充颜色同步改变。

3. 实验目的

(1) 掌握DOM事件的运用方法。
(2) 掌握DOM动态添加、删除HTML元素的方法。
(3) 掌握运用DOM CSS改变元素样式的方法。

4. 实验类型

综合设计。

5. 实验要求

(1) 脚本程序完全运用JavaScript程序实现。
(2) 用DOM添加删除元素显示客户区图形。

6. 实验环境

(1) 计算机:PC、内存8GB、主频1.8GHz及以上、硬盘500GB及以上。
(2) 操作系统:Windows XP、Windows 7、Windows 8、Windows 10。
(3) 开发环境:Visual Studio 2010或Adobe Dreamweaver。
(4) 浏览器:IE9及以上、Chrome、Firefox、Safari、Edge、QQ浏览器等。

7. 实验原理

DOM事件、CSS和添加删除元素直接相关的JavaScript语句,并辅以必要的说明与分析。

8. 遇到的问题及解决办法

(1) 给出所遇到的全部错误现象描述。
(2) 给出修正错误前设定的断点位置。
(3) 给出修正错误所监视的变量。
(4) 给出运行到断点处监视变量值。

9. 运行结果

给出所完成任务功能的屏幕截图。

第 6 章



jQuery

■ **知识目标**

- 掌握 jQuery 的安装方法
- 理解引入 jQuery 的必要性
- 掌握 jQuery 选择器与基本事件处理方法
- 掌握 jQuery 操纵 HTML、CSS 的技术
- 掌握 jQuery 动态添加、删除元素的方法
- 掌握 jQuery Ajax 与 ASP. NET 服务器进行交互的技术

■ **能力目标**

- 根据实际需求处理目标元素的适当事件
- 根据需要异步加载数据的元素，运用 Ajax 实现局部刷新
- 根据美工、布局和业务需求快速开发动态网页

■ **素质目标**

- 根据实际需求，运用 jQuery 编写简洁高效的脚本程序
- 根据应用需求恰当选择鼠标与键盘事件和相应的 jQuery 处理技术

■ **教学重点**

- jQuery HTML、CSS、按键解析、事件处理

■ **教学难点**

- 异步事件的跟踪调试

■ **建议学时**

- 理论：4 学时
- 实验：4 学时

6.1 概　　述

jQuery 是一个快速、简洁的 JavaScript 框架，是继 Prototype 之后又一个优秀的 JavaScript 框架。jQuery 设计的宗旨是 Write Less，Do More，即倡导写更少的代码，做

更多的事情。它封装 JavaScript 的常用功能代码，提供一种简便的 JavaScript 设计模式，优化 HTML 文档操作、事件处理、动画设计和 Ajax 交互。

jQuery 的核心特性概括为：具有独特的链式语法和短小清晰的多功能接口；具有高效灵活的 CSS 选择器，并且可对 CSS 选择器进行扩展；拥有便捷的插件扩展机制和丰富的插件。jQuery 兼容各种主流浏览器，如 IE 6.0 以上版本、Safari 2.0 以上版本、Opera 9.0 以上版本。

1. 编程开发

运行 jQuery 所需的条件很简单：一台计算机、一个智能电话或一个可以运行现代浏览器的设备。jQuery 对浏览器的要求也相当自由。官方网站列出了下列支持 jQuery 的浏览器：Firefox 2.0 以上版本、IE 6.0 以上版本、Safari 3.0 以上版本、Chrome 8.0 以上版本。

2. 语言特点

(1) 快速获取文档元素

jQuery 的选择机制构建于 CSS 选择器，提供了快速查询 DOM 文档中元素的功能，而且极大强化了 JavaScript 中获取页面元素的方式。

(2) 提供漂亮的页面动态效果

jQuery 中内置了一系列动画效果，可以开发出非常漂亮的网页，许多网站都使用 jQuery 的内置效果，比如淡入淡出、元素移除等动态特效。

(3) 创建 Ajax 局部刷新网页

Ajax 是异步 JavaScript 和 XML 的简称，可以开发出动态局部刷新网页。特别是开发服务器端网页时，比如 PHP 网站，需要频繁地与服务器通信，如果不使用 Ajax，每次数据更新不得不重新刷新网页，而使用 Ajax 特效后，可以对页面进行局部刷新，提供动态效果。

(4) 对 JavaScript 语言结构的增强

jQuery 提供了对基本 JavaScript 结构的增强，比如元素迭代和数组处理等操作。

(5) 增强的事件处理

jQuery 提供了各种页面事件，可以避免程序员在 HTML 中添加大量事件处理代码，而且其事件处理器消除了各种浏览器兼容性问题。

(6) 更改网页内容

jQuery 可以修改网页中的内容，比如更改网页的文本、插入或者翻转网页图像，jQuery 简化了原本使用 JavaScript 代码需要处理的方式。

3. 语言基础

(1) 选择器

jQuery 选择器允许对 HTML 元素组或单个元素进行操作，基于元素的 id、类、类型、属性、属性值等“查找”(或选择)HTML 元素。它基于已经存在的 CSS 选择器，除此之

外,还有一些自定义的选择器。jQuery 中的所有选择器都以美元符号开头:$()。

① 元素选择器。

jQuery 元素选择器基于元素名选取元素。如$("p")用于在页面中选取所有 <p> 元素。

② id 选择器。

jQuery #id 选择器通过 HTML 元素的 id 属性选取指定的元素。页面中元素的 id 应该是唯一的,所以要在页面中选取唯一的元素,需要通过 #id 选择器。通过 id 选取元素语法的语句如下:

```
$("#test")
```

③ class 选择器。

jQuery 类选择器可以通过指定的 class 查找元素。语法如下:

```
$(".test")
```

(2) 事件处理

在 jQuery 中,大多数 DOM 事件都有一个等效的 jQuery 方法。

在页面中指定一个单击事件:

```
$("p").click();
```

下面通过一个函数实现事件响应动作:

```
$("p").click(function(){ });
```

4. 技术应用

(1) 网站

只需要少量代码,即可将它们集成到网站上,并且能够帮助访问者分享网站上的内容。

(2) 移动端

jQuery Mobile 1.2 是 jQuery 运行在手机和平板设备上的版本。jQuery Mobile 1.2 为主流移动平台提供了 jQuery 的核心库,发布了一个完整统一的 jQuery 移动 UI 设计框架,在不同的智能手机和桌面电脑的 Web 浏览器上形成统一的用户 UI。支持全球主流的移动平台,对每个平台的支持分为 A、B、C 3 个等级,实现了对 Android 2.1～2.3、3.2、4.0、4.1,Windows Phone 7～7.5,Palm WebOS 1.4～2.0、3.0,Firefox Mobile 15,Opera Mobile 11.5～12 等平台的 A 级支持。jQuery Mobile 1. 2 的核心使得基本的 HTML 标签在所有的浏览器中生效,并且对网页的行为和效果均进行了增强,使网页在等级较高的浏览器中能获得优秀的体验,在较差的浏览器中也能正常使用。

6.2 基 础 知 识

6.2.1 jQuery 的安装

可以通过以下方法在网页中添加 jQuery。

(1) 从 jquery.com 下载 jQuery 库。

(2) 从 CDN 中载入 jQuery,如从 Google 中加载 jQuery。

有两个版本的 jQuery 可供下载。

(1) Production version:用于实际的网站中,已被精简和压缩。

(2) Development version:用于测试和开发(未压缩,是可读的代码)。

以上两个版本都可以从 jquery.com 中下载。

jQuery 库是一个 JavaScript 文件,使用 HTML 的<script>标签来引用,如下所示。

```
<head>
    <script src="jquery-1.10.2.min.js"></script>
</head>
```

将下载的文件放在网站内的一个目录下,也可以使用 jQuery。

百度、又拍云、新浪、谷歌和微软的服务器都存有 jQuery。如果站点是国内用户,建议使用百度、又拍云、新浪等国内 CDN 地址。许多用户访问其他站点时,已经从百度、又拍云、新浪、谷歌或微软加载过 jQuery。当他们再次访问这些站点时,会从缓存中加载 jQuery,以减少加载时间。同时,大多数 CDN 都可以确保当用户向其请求文件时,会从离用户最近的服务器上返回响应,这样也可以提高加载速度。否则,如果站点是国外的,可以使用谷歌和微软。

如需从菜鸟教程、又拍云、新浪、谷歌或微软引用 jQuery,请使用以下代码之一。

① 菜鸟教程 CDN。

```
<head>
    <script src="http://cdn.static.runoob.com/libs/jquery/1.10.2/jquery.min.js"></
    script>
</head>
```

② 百度 CDN。

```
<head>
    <script src="https://apps.bdimg.com/libs/jquery/2.1.4/jquery.min.js"></script>
</head>
```

③ 又拍云 CDN。

```
<head>
    <script src="http://upcdn.b0.upaiyun.com/libs/jquery/jquery-2.0.2.min.js"></
    script>
```

```
</head>
```

④ 新浪 CDN。

```
<head>
    <script src="http://lib.sinaapp.com/js/jquery/2.0.2/jquery-2.0.2.min.js"></
    script>
</head>
```

⑤ 谷歌 CDN。

```
<head>
    <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></
    script>
</head>
```

⑥ 微软 CDN。

```
<head>
    <script src="http://ajax.htmlnetcdn.com/ajax/jQuery/jquery-1.10.2.min.js"></
    script>
</head>
```

6.2.2 jQuery 语法

使用 jQuery 可以选取 HTML 元素，并对它们执行“操作”。jQuery 的语法功能是选取 HTML 元素，并对选取的元素执行某些操作。其通用格式如下：

```
$(selector).action();
```

用美元符号定义 jQuery。用选择符(selector)“查询”和“查找”HTML 元素。用 action()执行对元素的操作。例如：

```
$(document).ready(function(){
    $(this).hide();                 //隐藏当前元素;
    $("p").hide();                  //隐藏所有<p>元素;
    $("p.test").hide();             //隐藏所有 class="test"的<p>元素;
    $("#test").hide();              //隐藏所有 id="test" 的元素
});
```

以上实例中的所有 jQuery 函数位于 document ready 函数中。以防止文档在完全加载(就绪)之前运行 jQuery 代码。如果在文档没有完全加载之前就运行函数，操作可能失败。

6.2.3 jQuery 选择器

jQuery 选择器选择需要操作的 HTML 元素组或单个元素，基于元素的 id、类、类型、属性、属性值等“查找”(或选择)HTML 元素。它基于已经存在的 CSS 选择器，除此之

外，还有一些自定义的选择器。jQuery 中的所有选择器都以美元符号开头：$()。

1. 元素选择器

jQuery 元素选择器基于元素名选取元素。如用户单击按钮后，所有<p>元素都隐藏。

```
$(document).ready(function(){
    $("button").click(function(){
        $("p").hide();
    });
});
```

2. #id 选择器

jQuery #id 选择器选取特定 HTML 元素的 id 属性值的元素。页面中元素的 id 应该是唯一的，所以要在页面中选取唯一的元素，需要通过#id 选择器。如当用户单击按钮后，有 id="test"属性的元素将被隐藏。

```
$(document).ready(function(){
    $("button").click(function(){
        $("#test").hide();
    });
});
```

3. .class 选择器

jQuery 类选择器查找指定 class 属性值的元素。如用户单击按钮后，所有带 class="test" 属性的元素都隐藏。

```
$(document).ready(function(){
    $("button").click(function(){
        $(".test").hide();
    });
});
```

4. 更多实例

jQuery 的更多选择器实例如表 6.1 所示。

表 6.1 jQuery 的更多选择器实例

语　法	描　述
$("*")	选取所有元素
$(this)	选取当前 HTML 元素

续表

语　　法	描　　述
$("p.intro")	选取 class 为 intro 的<p>元素
$("p:first")	选取第一个<p>元素
$("ul li:first")	选取第一个 元素的第一个 元素
$("ul li:first-child")	选取每个 元素的第一个 元素
$("[href]")	选取带有 href 属性的元素
$("a[target='_blank']")	选取所有 target 属性值等于"_blank"的<a>元素
$("a[target!='_blank']")	选取所有 target 属性值不等于"_blank"的<a>元素
$(":button")	选取所有 type="button"的<input>元素和<button>元素
$("tr:even")	选取偶数位置的<tr>元素
$("tr:odd")	选取奇数位置的<tr>元素

6.2.4　jQuery 事件

jQuery 是为事件处理特别设计的。页面对不同访问者的响应叫做事件。事件处理程序指的是当 HTML 中发生某些事件时所调用的方法。在元素上移动鼠标、选取单选按钮、单击元素都属于事件。

在事件中常使用术语"触发"。例如，当用户按下按键时触发 keypress 事件。常见 DOM 事件如表 6.2 所示。

表 6.2　常见 DOM 事件

鼠标事件	键盘事件	表单事件	文档/窗口事件
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload
mouseover			
mouseout			
mousedown			
mouseup			
mousemove			

在 jQuery 中，大多数 DOM 事件都有一个等效的 jQuery 方法。页面中指定一个单击事件，语法如下：

```
$("p").click();
```

下一步是定义触发事件后的具体处理，通过以下事件函数实现：

```
$("p").click(function(){
    //动作触发后执行的代码!!
});
```

1. 文档就绪事件

$(document).ready() 方法定义在文档完全加载完后执行的函数。该事件方法在 jQuery 语法章节中已经提过。

2. 单击元素事件

单击元素时，会发生 click 事件。click()方法是当按钮单击事件被触发时调用的一个函数。该函数在单击 HTML 元素时执行。如当单击事件在某个<p>元素上触发时隐藏当前的<p>元素。

```
$("p").click(function(){
    $(this).hide();
});
```

3. 双击元素事件

双击元素时，会发生 dblclick 事件。dblclick()方法触发 dblclick 事件，或规定当发生 dblclick 事件时运行的函数。例如：

```
$("p").click(function(){
    $(this).hide();
});
```

4. 鼠标穿过元素事件

鼠标穿过元素时，会发生 mouseenter 事件。mouseenter() 方法触发 mouseenter 事件，或规定发生 mouseenter 事件时运行的函数。例如：

```
$("#p1").mouseenter(function(){
    alert('您的鼠标移到了 id="p1" 的元素上!');
});
```

5. 鼠标离开元素事件

鼠标指针离开元素时，会发生 mouseleave 事件。mouseleave()方法触发 mouseleave 事件，或规定发生 mouseleave 事件时运行的函数。例如：

```
$("#p1").mouseleave(function(){
    alert("再见,您的鼠标离开了该段落。");
});
```

6. 鼠标按下事件

鼠标移动到元素上方并按下鼠标时，会发生 mousedown 事件。mousedown() 方法触发 mousedown 事件，或规定发生 mousedown 事件时运行的函数。例如：

```
$("#p1").mousedown(function(){
    alert("鼠标在该段落上按下!");
});
```

7. 鼠标松开事件

在元素上松开鼠标时，会发生 mouseup 事件。mouseup()方法触发 mouseup 事件，或规定发生 mouseup 事件时运行的函数。例如：

```
$("#p1").mouseup(function(){
    alert("鼠标在段落上松开。");
});
```

8. 鼠标悬停事件

hover()方法用于模拟光标悬停事件。鼠标移动到元素上时，会触发指定的第 1 个函数(mouseenter)。鼠标移出这个元素时，会触发指定的第 2 个函数(mouseleave)。例如：

```
$("#p1").hover(
    function(){ alert("你进入了 p1!"); },
    function(){ alert("拜拜! 现在你离开了 p1!"); }
);
```

9. 获得焦点事件

元素获得焦点时，发生 focus 事件。当单击选中元素或通过 Tab 键定位到元素时，该元素就会获得焦点。focus()方法触发 focus 事件，或规定发生 focus 事件时运行的函数。例如：

```
$("input").focus(function(){
    $(this).css("background-color","#cccccc");
});
```

10. 失去焦点事件

元素失去焦点时，发生 blur 事件。blur()方法触发 blur 事件，或规定发生 blur 事件时运行的函数。例如：

```
$("input").blur(function(){
    $(this).css("background-color","#ffffff");
});
```

6.3 jQuery HTML

6.3.1 读写内容与属性

jQuery 拥有操作 HTML 元素和属性的方法。操控 DOM 是 jQuery 的重要功能。jQuery 提供一系列与 DOM 相关的方法,这使访问和操作元素和属性变得很容易。

1. 读取内容

3 个简单实用的用于 DOM 操作的 jQuery 方法如下所示。

(1) text():设置或返回所选元素的文本内容。

(2) html():设置或返回所选元素的内容(包括 HTML 标记)。

(3) val():设置或返回表单字段的值。

下面的例子说明如何通过 jQuery text()和 html()方法来获得内容。

```
$("#btn1").click(function(){ alert("Text: "+$("#test").text()); });
$("#btn2").click(function(){ alert("HTML: "+$("#test").html()); });
```

而下面的例子演示如何通过 jQuery val() 方法获得输入字段的值。

```
$("#btn1").click(function(){ alert("值为: "+$("#test").val()); });
```

2. 读取属性

jQuery attr() 方法用于获取属性值。下面的例子说明如何获得链接中 href 属性的值。

```
$("button").click(function(){ alert($("#runoob").attr("href")); });
```

3. 设置内容

使用读取 3 个相同的方法来设置内容。

(1) text():设置或返回所选元素的文本内容。

(2) html():设置或返回所选元素的内容(包括 HTML 标记)。

(3) val():设置或返回表单字段的值。

下面的例子说明如何通过 text()、html()及 val()方法来设置内容。

```
$("#btn1").click(function(){ $("#test1").text("Hello world!");});
$("#btn2").click(function(){ $("#test2").html("<b>Hello world!</b>"); });
$("#btn3").click(function(){ $("#test3").val("RUNOOB"); });
```

4. 设置属性

jQuery attr()方法也用于设置/改变属性值。下面的例子说明如何改变(设置)链接

中 href 属性的值。

```
$("button").click(function(){$("#runoob").attr("href","http://www.runoob.com/jquery");});
```

attr()方法也允许同时设置多个属性。下面的例子演示如何同时设置 href 和 title 属性。

```
$("button").click(function(){
    $("#runoob").attr({
        "href" : "http://www.runoob.com/jquery",
        "title" : "jQuery 教程"
    });
});
```

6.3.2 添加删除元素

1. 添加元素

使用 jQuery 可以添加新元素。添加新元素的方法有如下 4 个。

① append():在被选元素内结尾插入。

② prepend():在被选元素内开头插入。

③ after():在被选元素之后插入。

④ before():在被选元素之前插入。

(1) append()方法

append()方法在被选元素的结尾插入新元素,如例 6-3-2-1 所示。

例 6-3-2-1 body 内结尾追加新内容。

① 源文档例 6-3-2-1.htm 内容。

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
        <script src="scripts/jquery-1.4.1.js" type="text/javascript">
        </script>
        <script type="text/javascript">
            function appendText() {
                var txt1="<p>Text.</p>";                    //以 HTML 创建新元素
                var txt2=$("<p></p>").text("Text.");        //以 jQuery 创建新元素
                var txt3=document.createElement("p");
                txt3.innerHTML="Text.";                     //通过 DOM 来创建文本
                $("body").append(txt1, txt2, txt3, txt1, "Hello","everyone");   //追加新内容
            }
        </script>
```

```
    </head>
    <body>
        <p>This is a paragraph.</p>
        <button onclick="appendText()">追加文本</button>
    </body>
</html>
```

② 浏览器显示结果。

浏览器显示结果如图 6.1 和图 6.2 所示。



图 6.1　例 6-3-2-1 的浏览器显示结果（未单击“追加文本”按钮）

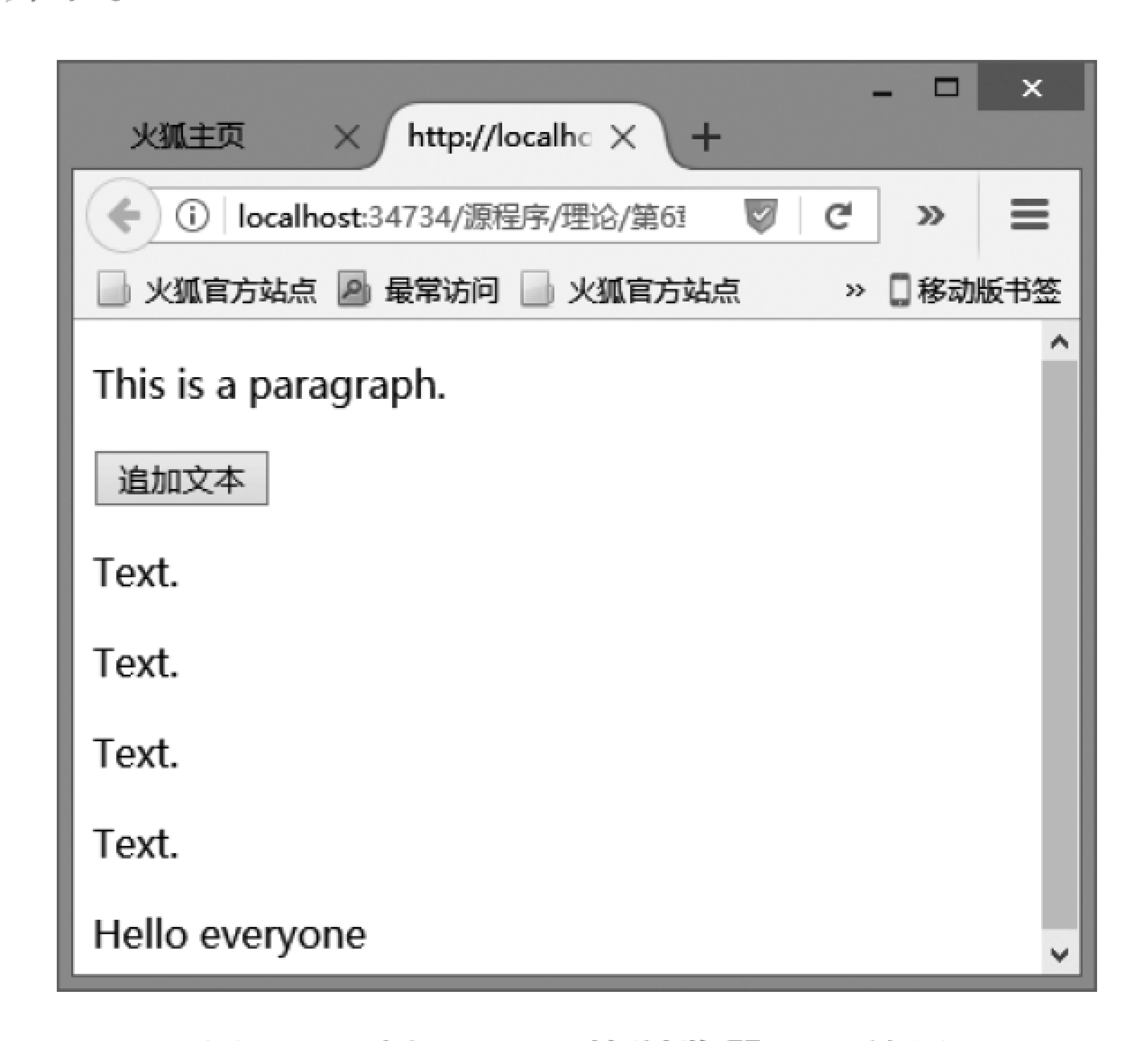


图 6.2　例 6-3-2-1 的浏览器显示结果（单击“追加文本”按钮后）

③ 在查看器中查看文档结构。

在查看器中查看文档结构如图 6.3 所示。

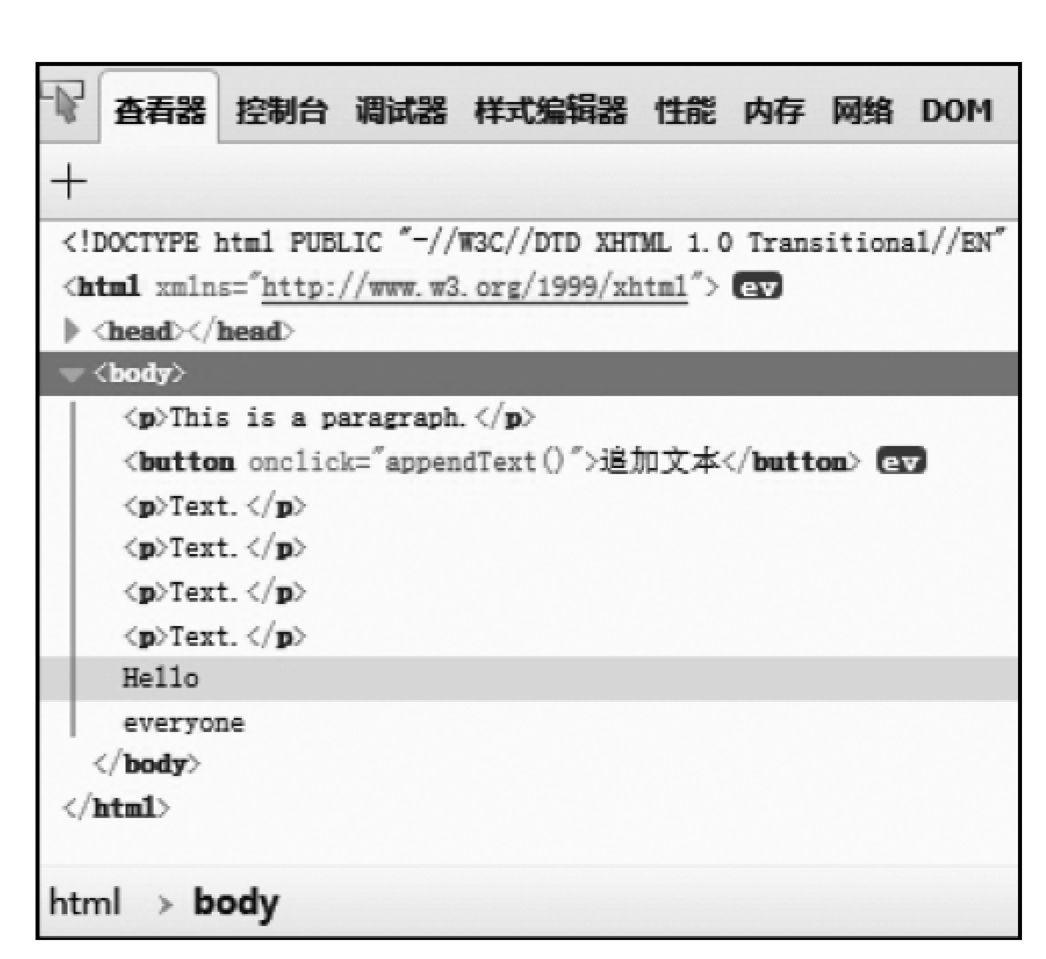


图 6.3　例 6-3-2-1 的浏览器中的查看器显示（单击“追加文本”按钮后）

(2) prepend()方法

prepend()方法在被选元素的开头插入元素,如例 6-3-2-2 所示。

例 6-3-2-2 body 内开头追加新元素。

① 源文档例 6-3-2-2.htm 文件内容。

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
        <script src="scripts/jquery-1.4.1.js" type="text/javascript">
        </script>
        <script type="text/javascript">
            function appendText() {
                var txt1="<p>Text.</p>";                    //以 HTML 创建新元素
                var txt2=$("<p></p>").text("Text.");        //以 jQuery 创建新元素
                var txt3=document.createElement("p");
                txt3.innerHTML="Text.";                     //通过 DOM 来创建文本
                $("body").prepend(txt1, txt2, txt3, txt1, "Hello",
                "everyone");                                //追加新内容
            }
        </script>
    </head>
    <body>
        <p>This is a paragraph.</p>
        <button onclick="appendText()">追加文本</button>
    </body>
</html>
```

② 浏览器显示结果。

未单击"追加文本"按钮时,显示结果与图 6.1 相同。单击"追加文本"按钮后,显示结果如图 6.4 所示。

③ 在查看器中查看文档结构。

在查看器中查看文档结构,如图 6.5 所示。

(3) after()方法

前面使用 append()和 prepend()方法添加新元素,是在被选元素的开头和结尾插入内容。而且,append()和 prepend()方法能够通过参数接收无限数量的新元素。after()方法在被选元素之后插入元素,如例 6-3-2-3 所示。

例 6-3-2-3 img 元素后添加新元素。



图 6.4 例 6-3-2-2 的浏览器显示结果（单击“追加文本”按钮后）

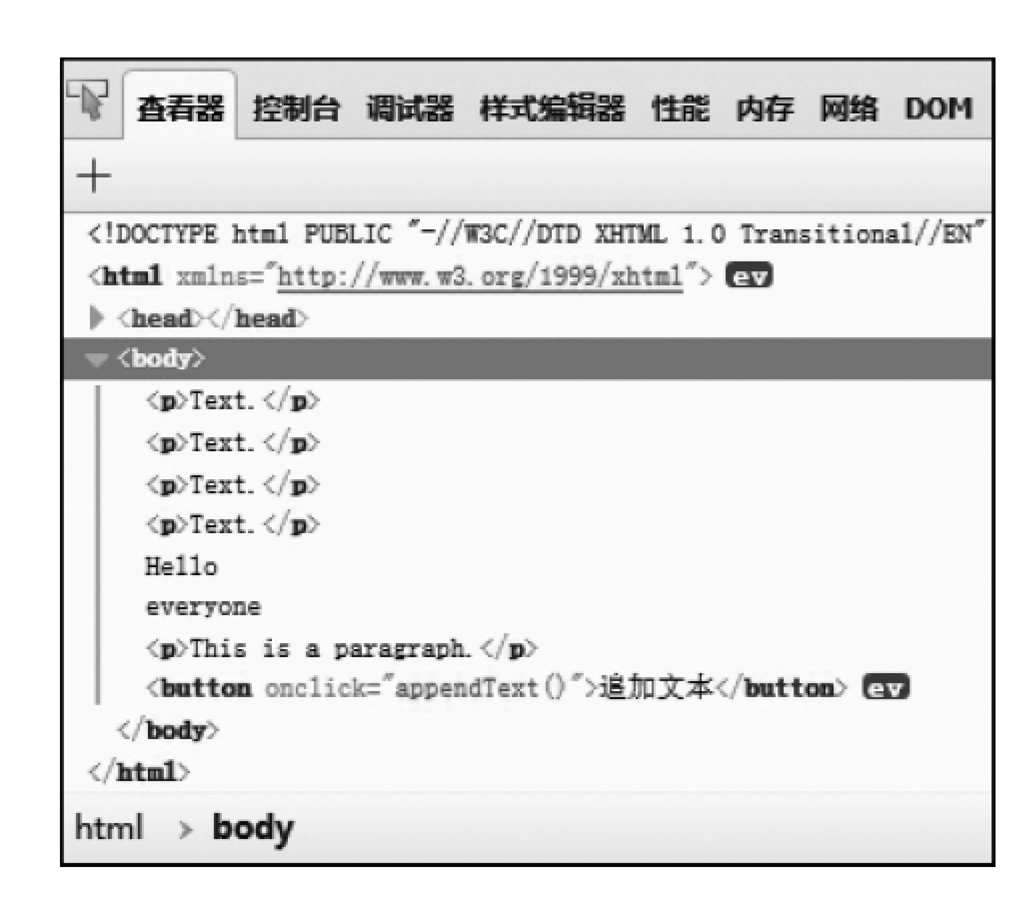


图 6.5 例 6-3-2-2 的浏览器中的查看器显示（单击“追加文本”按钮后）

① 源文档例 6-3-2-3.htm 文件内容。

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
        <script src="scripts/jquery-1.4.1.js" type="text/javascript">
        </script>
        <script type="text/javascript">
            function afterText() {
                var txt1="<b>I</b>";                          //以 HTML 创建元素
                var txt2=$("<i></i>").text("love ");          //通过 jQuery 创建元素
                var txt3=document.createElement("big");       //通过 DOM 创建元素
                txt3.innerHTML="jQuery!";
                $("img").after(txt1, txt2, txt3);                //在 img 之后插入新元素
            }
        </script>
    </head>
    <body>
        <img src="/Img/eg_w3school.gif" alt="W3School Logo" />
        <br/><br/>
        <button onclick="afterText()">在图片后面添加文本</button>
    </body>
</html>
```

② 浏览器显示结果。

浏览器显示结果如图 6.6 和图 6.7 所示。



图 6.6　例 6-3-2-3 的浏览器显示结果(未单击“在图片后插入文本”按钮)



图 6.7　例 6-3-2-3 的浏览器显示结果(单击“在图片后插入文本”按钮后)

③ 在查看器中查看文档结构。

在查看器中查看文档结构,如图 6.8 所示。



图 6.8　例 6-3-2-3 的浏览器中的查看器显示(单击“在图片后插入文本”按钮后)

2. 删除元素

删除元素一般使用以下两个 jQuery 方法。

- remove()：删除被选元素(及其子元素)。
- empty()：从被选元素中删除子元素。

(1) remove ()方法

remove()方法删除被选元素及其子元素。例如：

```
$("#div1").remove();
```

(2) empty()方法

empty()方法删除被选元素的子元素。例如：

```
$("#div1").empty();
```

remove()方法也可接受一个参数，允许对被删元素进行过滤。该参数可以是任何 jQuery 选择器的语法。以下例子删除 class="italic"的所有<p>元素。

```
$("p").remove(".italic");
```

6.3.3 操纵 CSS

1. 增删 CSS 类

使用 jQuery 可以很容易地操作 CSS 元素。jQuery 拥有如下 3 种操作 CSS 类的方法：

① addClass()：向被选元素添加一个或多个类。

② removeClass()：从被选元素删除一个或多个类。

③ toggleClass()：对被选元素进行添加/删除类的切换操作。

类的实例样式表如下：

```
.important { font-weight : bold; font-size : xx-large; }
.blue { color : blue; }
```

(1) addClass()方法

下面的例子展示向一个或多个不同的元素添加 class 属性。

```
$("button").click(function(){
    $("h1,h2,p").addClass("blue");
    $("div").addClass("important");
});
```

也可以在 addClass() 方法中规定多个类。

```
$("button").click(function(){ $("#div1").addClass("important blue"); });
```

（2）removeClass()方法

下面的例子说明如何在不同的元素中删除指定的 class 属性。

```
$("button").click(function(){ $("h1, h2, p").removeClass("blue"); });
```

（3）toggleClass()方法

下面的例子展示如何使用 toggleClass() 方法。该方法对被选元素进行添加/删除类的切换操作。

```
$("button").click(function(){ $("h1,h2,p").toggleClass("blue");});
```

2. 读写 CSS

读写 CSS 使用同一个 css()方法。css()方法可设置或返回被选元素的一个或多个样式属性。

（1）读取 CSS 属性

返回指定 CSS 属性的值的语法如下：

```
css("propertyname");
```

下面的例子将返回首个匹配元素的 background-color 值。

```
$("p").css("background-color");
```

（2）设置 CSS 属性值

设置指定 CSS 属性值的语法如下：

```
css("propertyname","value");
```

下面的例子将为所有匹配元素设置 background-color 值。

```
$("p").css("background-color","yellow");
```

同时设置多个 CSS 属性值的语法如下：

```
css({"propertyname":"value","propertyname":"value",...});
```

下面的例子将为所有匹配元素设置 background-color 和 font-size。

```
$("p").css({"background-color":"yellow","font-size":"200%"});
```

6.3.4 操纵尺寸

jQuery 提供以下 6 个处理尺寸的重要方法。

① width()：设置或返回元素的宽度(不包括内边距、边框或外边距)。

② height()：设置或返回元素的高度(不包括内边距、边框或外边距)。

③ innerWidth()：返回元素的宽度(包括内边距)。

④ innerHeight()：返回元素的高度(包括内边距)。

⑤ outerWidth()：返回元素的宽度(包括内边距和边框)。

⑥ outerHeight()：返回元素的高度(包括内边距和边框)。

例 6-3-4-1 说明了这 6 个函数的功能和使用方法。

例 6-3-4-1 浏览器显示 1 个 div 和 1 个按钮。div 内文本为 div 自身的尺寸信息。按钮功能是放大 div 的宽和高各 1 倍。单击按钮 1 次，放大当前尺寸 1 次。

① 源文档例 6-3-4-1.htm 文件内容。

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
        <script src="scripts/jquery-1.4.1.js" type="text/javascript">
        </script>
        <script type="text/javascript">
            function Size() {
                var txt="";
                txt+="div 的宽度是: "+$("div").width()+"</br>";
                txt+="div 的高度是: "+$("div").height()+"</br>"; ;
                txt+="div 宽度,包含内边距: "+$("div").innerWidth()+"</br>";
                txt+="div 高度,包含内边距: "+$("div").innerHeight()+"</br>";
                txt+="div 宽度,包含内边距和边框: "+$("div").outerWidth()+"</br>";
                txt+="div 高度,包含内边距和边框: "+$("div").outerHeight();
                $("div").html(txt);
            }
            function Scale() {
                $("div").width($("div").width() * 2);
                $("div").height($("div").height() * 2);
                Size();
            }
            $(document).ready(function () {
                Size();
                $("div").click(Scale); });
        </script>
    </head>
    <body>
        <div style="border:1px solid black; width:300px; height:150px;
        padding:20px; margin:10px;"></div>
        <button onclick="Scale()">宽和高各放大一倍</button>
    </body>
</html>
```

② 浏览器显示结果。

浏览器显示结果分别如图 6.9 和图 6.10 所示。



图 6.9　例 6-3-4-1 的浏览器显示结果(未单击"宽和高各放大一倍"按钮)



图 6.10　例 6-3-4-1 的浏览器显示结果(单击"宽和高各放大一倍"按钮 1 次)

6.4　遍　　历

遍历是根据元素相对于其他元素的关系来"查找"(或选取)HTML 元素。以某项选择开始，并沿着该选择移动，直到找到目标元素为止。

图 6.11 展示了一个家族树，是本小节所有例子所用的文档树实例。通过 jQuery 遍历，能够从被选(当前的)元素开始，在家族树中向上移动(祖先)，向下移动(子孙)，水平移动(同胞)。这种移动称为对 DOM 进行遍历。

<div>元素是<ul>的父元素，同时是其中所有内容的祖先。<ul>元素是<li>元素的父元素，同时是<div>的子元素。左边的<li>元素是<span>的父元素以及

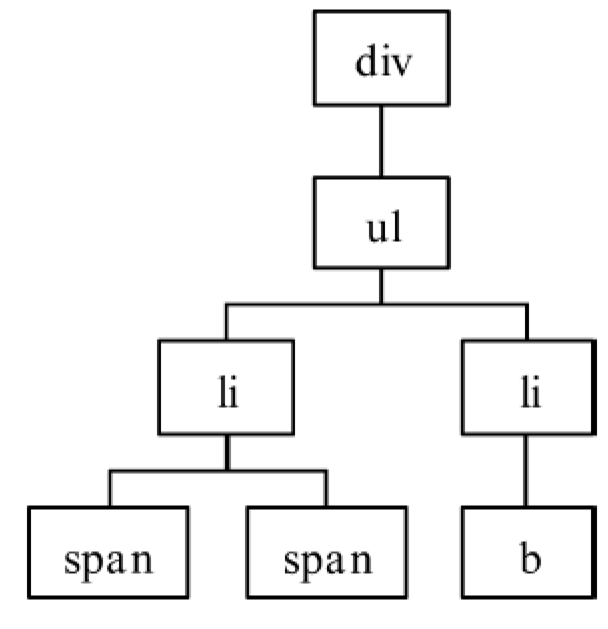


图 6.11　一个家族树示意图

的子元素，同时是<div> 的后代。<span>元素是<li>的子元素，同时是<ul>和<div>的后代。两个<li>元素是同胞(拥有相同的父元素)。右边的<li>元素是<b>的父元素，<ul>的子元素，同时是<div>的后代。<b>元素是右边的<li>的子元素，同时是<ul>和<div>的后代。祖先是父、祖父、曾祖父等。后代是子、孙、曾孙等。同胞拥有相同的父。

1. 祖先

祖先是父、祖父或曾祖父等。通过祖先，向上遍历 DOM 树，以查找目标元素。用于向上遍历 DOM 树的方法如下：

① parent()：返回被选元素的直接父元素。

② parents()：返回被选元素的所有祖先元素，它一路向上直到文档的根元素(<html>)。

③ parentsUntil()：返回介于两个给定元素之间的所有祖先元素。

(1) parent()方法

该方法只会向上一级对 DOM 树进行遍历。下面的例子返回每个<span>元素的直接父元素。

```
$(document).ready(function(){ $("span").parent(); });
```

(2) parents()方法

下面的例子返回所有<span>元素的所有祖先。

```
$(document).ready(function(){ $("span").parents(); });
```

也可以使用可选参数来过滤对祖先元素的搜索。下面的例子返回所有<span>元素的所有祖先，并且它是<ul>元素。

```
$(document).ready(function(){ $("span").parents("ul"); });
```

(3) parentsUntil()方法

下面的例子返回介于<span>与<div>元素之间的所有祖先元素。

```
$(document).ready(function(){ $("span").parentsUntil("div"); });
```

2. 后代

后代是子、孙、曾孙等。向下遍历 DOM 树，以查找元素的后代，方法如下。

① children()：返回被选元素的所有直接子元素。

② find()：返回被选元素的后代元素，一路向下直到最后一个后代。

(1) children()方法

下面的例子返回每个<div>元素的所有直接子元素。

```
$(document).ready(function(){ $("div").children(); });
```

也可以使用可选参数来过滤对子元素的搜索。下面的例子返回类名为“1”的所有

<p>元素,并且它们是<div>的直接子元素。

```
$(document).ready(function(){ $("div").children("p.1");});
```

(2) find()方法

下面的例子返回属于<div>后代的所有<span>元素。

```
$(document).ready(function(){ $("div").find("span"); });
```

下面的例子返回<div>的所有后代。

```
$(document).ready(function(){ $("div").find("*"); });
```

3. 同胞

同胞拥有相同的父元素。在 DOM 树中遍历元素的同胞元素方法如下:

① siblings():返回被选元素的所有同胞元素。
② next():返回被选元素的下一个同胞元素。
③ nextAll():返回被选元素的所有跟随的同胞元素。
④ nextUntil():返回介于两个给定参数之间的所有跟随的同胞元素。
⑤ prev():返回被选元素的前一个同胞元素。
⑥ prevAll():返回被选元素的所有前面的同胞元素。
⑦ prevUntil():返回介于两个给定参数之间的所有前面的同胞元素。

(1) siblings()方法

下面的例子返回<h2>的所有同胞元素。

```
$(document).ready(function(){ $("h2").siblings(); });
```

也可以使用可选参数来过滤对同胞元素的搜索。下面的例子返回属于<h2>的同胞元素的所有<p>元素。

```
$(document).ready(function(){ $("h2").siblings("p"); });
```

(2) next()方法

该方法只返回一个元素。下面的例子返回<h2>的下一个同胞元素。

```
$(document).ready(function(){ $("h2").next(); });
```

(3) nextAll()方法

下面的例子返回<h2>的所有跟随的同胞元素。

```
$(document).ready(function(){ $("h2").nextAll(); });
```

(4) nextUntil()方法

下面的例子返回介于<h2>与<h6>元素之间的所有同胞元素。

```
$(document).ready(function(){ $("h2").nextUntil("h6"); });
```

(5) prev()、prevAll()及 prevUntil()方法

prev()、prevAll()及 prevUntil()方法的工作方式与上面的方法类似，只不过方向相反，它们返回的是前面的同胞元素。

4. 过滤

缩小搜索元素的范围有 3 个最基本的过滤方法：first()、last()和 eq()，它们基于其在一组元素中的位置来选择一个特定的元素。其他过滤方法，比如 filter()和 not()，选取匹配或不匹配某项指定标准的元素。

(1) first()方法

first()方法返回被选元素的首个元素。下面的例子选取首个<div>元素内部的第一个<p>元素。

```
$(document).ready(function(){ $("div p").first(); });
```

(2) last()方法

last()方法返回被选元素的最后一个元素。下面的例子选择最后一个<div>元素中的最后一个<p>元素。

```
$(document).ready(function(){ $("div p").last(); });
```

(3) eq()方法

eq()方法返回被选元素中带有指定索引号的元素。索引号从 0 开始，因此首个元素的索引号是 0 而不是 1。下面的例子选取第二个<p>元素(索引号 1)。

```
$(document).ready(function(){  $("p").eq(1); });
```

(4) filter()方法

filter()方法允许规定一个标准进行筛选。不匹配这个标准的元素会从集合中删除，匹配的元素会返回。下面的例子返回带有类名“url”的所有<p>元素。

```
$(document).ready(function(){ $("p").filter(".url"); });
```

(5) not()方法

not()方法与 filter()相反，返回不匹配标准的所有元素。下面的例子返回不带有类名“url”的所有<p>元素。

```
$(document).ready(function(){ $("p").not(".url"); });
```

6.5 Ajax

Ajax 是与服务器交换数据的技术，它在不重载全部页面的情况下实现对部分网页的更新。Ajax，即异步 JavaScript 和 XML(Asynchronous JavaScript and XML)，它不重载整个网页，而是在后台加载数据，并在网页上显示。使用 Ajax 的应用程序案例有谷歌地图、腾讯微博、优酷视频、人人网等。

使用jQuery Ajax方法，能够使用HTTP GET和HTTP POST，从远程服务器上请求文本、HTML、XML或JSON，能同时把这些外部数据直接载入网页的被选元素中。如果没有jQuery，Ajax编程会有些难度。

编写常规的Ajax代码并不容易，因为不同的浏览器对Ajax的实现并不相同。这意味着必须编写额外的代码对浏览器进行测试。不过，jQuery团队解决了这个难题，只需要一行简单的代码，就可以实现Ajax功能。

1. load()方法

load()方法是简单但强大的Ajax方法。它从服务器加载数据，并把返回的数据放入被选元素中。语法如下：

```
$(selector).load(URL, data, callback);
```

URL是必需的参数，规定被加载数据的URL。data参数可选，规定与请求一同发送的查询字符串键/值对集合。callback参数可选，是load()方法完成后所执行的函数名称。

例6-5-1-1 把文件“demo_test.txt”的内容加载到指定的<div>元素。

(1) 例6-5-1-1-demo_test.txt文件

例6-5-1-1-demo_test.txt文件内容如下。

```
<h2>jQuery AJAX是个非常棒的功能！</h2>
<p id="p1">这是段落的一些文本。</p>
```

(2) 例6-5-1-1.htm文件

例6-5-1-1.htm源文档如下。

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
        <script src="scripts/jquery-1.4.1.js" type="text/javascript">
        </script>
        <script>
            $(document).ready(function () { $("div").load("例6-5-1-1-demo_
            test.txt"); });
        </script>
    </head>
    <body>
        <div></div>
    </body>
</html>
```

也可以把jQuery选择器添加到URL参数。下面的例子把demo_test.txt文件中id="p1"的元素的内容加载到指定的<div>元素中：

```
$("#div1").load("例 6-5-1-demo_test.txt #p1");
```

可选的 callback 参数规定当 load()方法完成后所要允许的回调函数。回调函数可以设置不同的参数。

① responseTxt：包含调用成功时的结果内容。

② statusTXT：包含调用的状态。

③ xhr：包含 XMLHttpRequest 对象。

例 6-5-1-2 在例 6-5-1-1 的基础上，load()方法完成后显示一个提示框。如果 load()方法已成功，则显示“外部内容加载成功！”，如果失败，则显示错误消息。

(1) 例 6-5-1-2.htm 文件

源文档例 6-5-1-2.htm 文件内容如下。

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
        <script src="scripts/jquery-1.4.1.js" type="text/javascript">
            </script>
        <script>
            $(document).ready(function(){
                $("button").click(function(){
                    $("div").load("例 6-5-1-2-demo_test.txt", function
                    (responseTxt, statusTxt, xhr) {
                        if (statusTxt=="success")
                            alert("外部内容加载成功!");
                        if (statusTxt=="error")
                            alert("Error: "+xhr.status+": "+xhr.statusText);
                    });
                });
            });
        </script>
    </head>
    <body>
        <button type="button" style="height:30px;width:150px;">从服务器加载数据</
        button>
        <div></div>
    </body>
</html>
```

(2) 浏览器显示结果

浏览器显示结果如图 6.12 和图 6.13 所示。

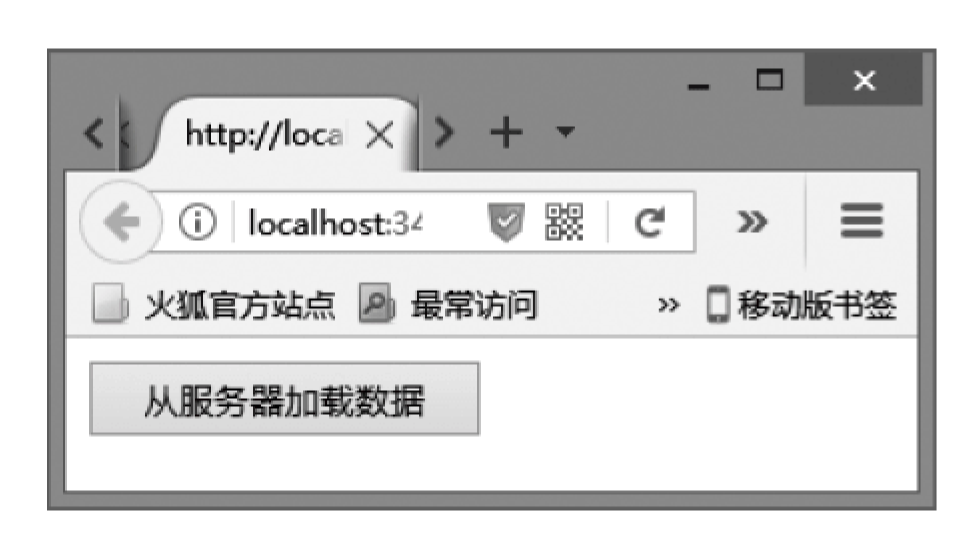


图 6.12　未单击“从服务器加载数据”按钮的网页



图 6.13　单击“从服务器加载数据”按钮的网页

2. $.get()方法

通过 HTTP GET 请求从服务器上请求数据。语法如下：

```
$.get(URL, callback);
```

URL 参数：必需参数，规定所请求的 URL。

callback 参数：可选参数，规定请求成功后所执行的函数名。

例 6-5-1-3　使用 $.get()方法从 ASP.NET 服务器上的一个文件中取回数据。单击网页的“从服务器加载数据”按钮，则执行 $.get()方法，从指定的服务器文件获取文本数据，用 alert 函数显示响应数据。

(1) 浏览器端

例 6-5-1-3.htm 文件为浏览器端源文档，如下所示。

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
        <script src="scripts/jquery-1.4.1.js" type="text/javascript">
        </script>
        <script>
            $(document).ready(function(){
                $("button").click(function () {
                    $.get("例 6-5-1-3.aspx", function (data, status) {
                        alert("数据: "+data+"\n状态: "+status);
                    });
                });
            });
        </script>
    </head>
    <body>
```

```
        <button type="button" style="height:30px;width:150px;">从服务器加载数据</
        button>
    </body>
</html>
```

(2) 服务器端

例 6-5-1-3.aspx 文件为服务器端源文件,内容如下。

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="例 6-5-1-3.aspx.cs" Inherits=
"理论_第 6 章_jQuery_例 6_5_1_3" %>
<%Response.Write("hello, this is response from ASP.NET"); %>
```

(3) 浏览器显示

未单击按钮的网页如图 6.12 所示,而单击按钮的网页如图 6.14 所示。



图 6.14　单击“从服务器加载数据”按钮的网页

3. $.post()方法

$.post()方法通过 HTTP POST 请求从服务器上请求数据。语法如下:

```
$.post(URL, data, callback);
```

URL 参数:必需参数,规定所请求的 URL。

data 参数:可选参数,规定连同请求发送的数据。

callback 参数:可选参数,规定请求成功后所执行的函数名。

例 6-5-1-4　使用 $.post()连同请求一起发送数据到 ASP.NET 服务器。单击网页的“从服务器加载数据”按钮,则执行 $.post()方法,向指定的 aspx 网页传送数据,并从指定的服务器文件获取文本数据,用 alert 函数显示响应数据。

(1) 浏览器端

例 6-5-1-4.htm 文件为浏览器端源文档,如下所示。

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
```

```
        <title></title>
        <script src="scripts/jquery-1.4.1.js" type="text/javascript"></script>
        <script>
            $(document).ready(function(){
                $("button").click(function () {
                    $.post("例 6-5-1-4.aspx",
                         { name: "Donald Duck", city: "Duckburg" },
                          function (data, status) { alert("数据: "+data+"\n 状态: "+
                         status); });
                    });
            });
        </script>
    </head>
    <body>
        <button type="button" style="height:30px;width:150px;">从服务器加载数据</
        button>
    </body>
</html>
```

(2) 服务器端

例 6-5-1-4.aspx 文件为服务器端源文件,内容如下。

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="例 6-5-1-4.aspx.cs" Inherits=
"理论_第 6 章_jQuery_例 6_5_1_4" %>
<%string fname,city;
fname=Request.Form["name"];
city=Request.Form["city"];
Response.Write("Dear "+fname+". ");
Response.Write("Hope you live well in "+city+".");
    %>
```

(3) 浏览器显示

未单击“从服务器加载数据”按钮的网页如图 6.12 所示,而单击“从服务器加载数据”按钮的网页如图 6.15 所示。

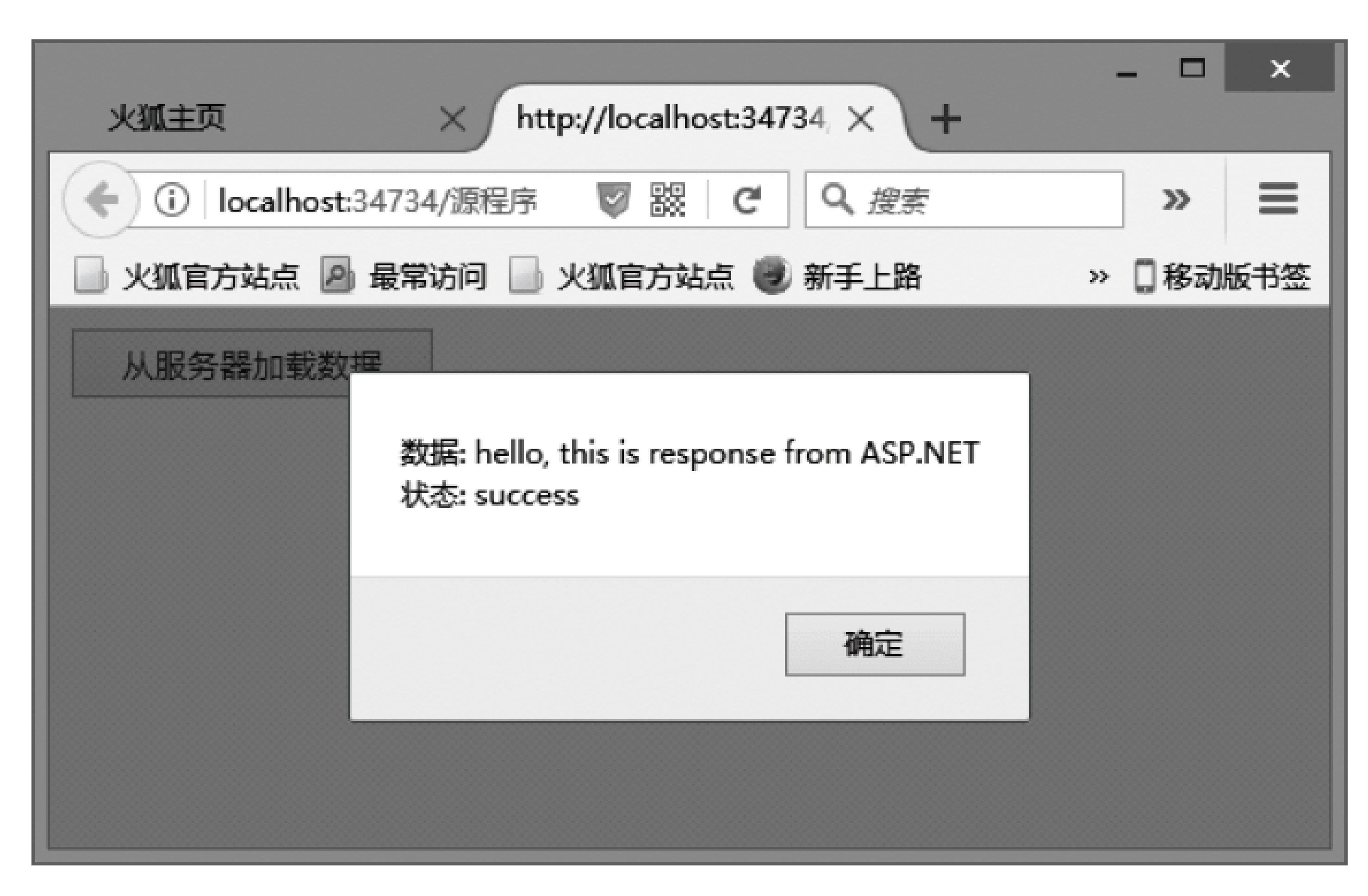


图 6.15　单击“从服务器加载数据”按钮的网页

6.6 实验任务

1. 实验题目

jQuery 脚本程序设计。

2. 程序功能

设计一个画图网页，含横向水平主菜单、纵向树形面板和客户区。横向水平主菜单含有长方形、圆、菱形、正方形、椭圆 5 个菜单项。纵向树形面板含有线型、颜色、动静 3 个选项，线型含有实线、虚线 2 个子项，颜色含有红、绿、蓝 3 个子项，动静含有动和静 2 个子选项。

单击长方形，客户区显示长方形。单击正方形客户区，显示正方形。单击线型，当前图形线型改变。单击颜色，当前图形的边框和填充颜色同步改变。单击动静的动选项，当前正方形水平匀速运动；单击动静的静选项，如果当前正方形正在运动，则在当前位置停下。

按下 B 键，当前图形边框设置为蓝色；按下 R 键，当前图形边框设置为红色。

3. 实验目的

(1) 掌握 jQuery 事件的运用方法。

(2) 掌握 jQuery 动态添加删除 HTML 元素的方法。

(3) 掌握运用 jQuery CSS 改变元素样式的方法。

4. 实验类型

综合设计。

5. 实验要求

(1) 脚本程序完全运用 jQuery 程序实现。

(2) 用 jQuery 添加删除元素，显示客户区图形。

(3) 所有 jQuery 事件绑定脚本程序置于文档就绪事件中。

(4) 所有 jQuery 脚本程序置于 head 元素中。

6. 实验环境

(1) 计算机：PC、内存 8GB、主频 1.8GHz 及以上、硬盘 500GB 及以上。

(2) 操作系统：Windows XP、Windows 7、Windows 8、Windows 10。

(3) 开发环境：Visual Studio 2010。

(4) 浏览器：IE9 及以上、Chrome、Firefox、Safari、Edge、QQ 浏览器等。

7. 实验原理

事件、CSS添加删除元素直接相关的jQuery语句,并辅以必要的说明与分析。

8. 遇到的问题及解决办法

(1) 给出所遇到的全部错误现象描述。
(2) 给出修正错误前设定的断点位置。
(3) 给出修正错误所监视的变量。
(4) 给出运行到断点处的监视变量值。

9. 运行结果

给出所完成任务功能的屏幕截图。

第 7 章 chapter 7

JSON

■ **知识目标**

- 掌握动画机制的实现方法
- 理解 JSON 的本质
- 掌握 JSON 的面向对象知识

■ **能力目标**

根据实际需求，运用 JSON 进行面向对象的程序开发

- 根据实际情况恰当运用动画机制
- 根据要求熟练应用 jQuery 的 CSS 和事件处理
- 根据实际需求恰当选用定位方式

■ **素质目标**

- 以面向对象的视角观察、分析、设计程序

■ **教学重点**

- 运用 JSON 设计面向对象的脚本程序

■ **教学难点**

- 识别和封装对象

■ **建议学时**

- 理论：2 学时
- 实验：2 学时

7.1 JSON 基础

JSON(JavaScript Object Notation)是一种轻量级的数据交换格式。它是基于 ECMAScript 的一个子集，采用完全独立于语言的文本格式，沿用类似 C 语言家族的习惯(包括 C、C++、C#、Java、JavaScript、Perl、Python 等)。这些特性使 JSON 成为理想的数据交换语言。它易于人阅读和编写，同时也易于机器解析和生成(一般用于提升网络传输速率)。

JSON 类似于 XML。它是纯文本，具有"自我描述性"，具有层级结构（值中存在值），可通过 JavaScript 进行解析，JSON 格式数据可使用 Ajax 进行传输。

JSON 与 XML 也有不同之处。它没有结束标签，更简短，读写的速度更快，使用内建的 JavaScript eval()方法进行解析，使用数组和保留字。

对于 Ajax 应用程序来说，JSON 比 XML 更快、更易使用。使用 XML 有 3 个步骤，首先读取 XML 文档，而后使用 XML DOM 来循环遍历文档，最后读取值并存储在变量中。使用 JSON 只有 2 步，读取 JSON 字符串和用 eval()处理 JSON 字符串。

7.1.1 语法

JSON 语法是 JavaScript 对象表示语法的子集，基本规则如下。

① 数据在键值对中。

② 数据由逗号分隔。

③ 花括号保存对象。

④ 方括号保存数组。

1. JSON 名称/值对

JSON 数据的书写格式是：名称/值对。

名称/值对组合中的名称写在前面（在双引号中），值对写在后面（同样在双引号中），中间用冒号隔开，例如：

```
"firstName":"John"
```

等价于 JavaScript 语句：

```
firstName="John"
```

2. JSON 值

JSON 值有如下 5 种。

(1) 数字（整数或浮点数）。

(2) 字符串（在双引号中）。

(3) 逻辑值（true 或 false）。

(4) 数组（在方括号中）。

(5) 对象（在花括号中）。

(6) null。

3. 基础结构

JSON 结构有对象和数组 2 种基本结构。通过这两种结构可以表示各种复杂的结构。

(1) 对象

对象在 JavaScript 中表示为"{}"括起来的内容，数据结构为 {key：value，key：

value, ...}的键值对的结构,在面向对象的语言中,key 为对象的属性,value 为对应的属性值,所以很容易理解。取值方法为 对象.key,以获取属性值,这个属性值的类型可以是数字、字符串、数组、对象。

(2) 数组

数组在 JavaScript 中是中括号"[]"括起来的内容,数据结构为["java", "javascript", "vb", ...],取值方式和所有语言中一样,使用索引获取,字段值的类型可以是数字、字符串、数组、对象几种。

7.1.2 示例

JSON 可以将 JavaScript 对象中表示的一组数据转换为字符串,然后在函数之间传递这个字符串,或者在异步应用程序中将字符串从 Web 客户机传递给服务器端程序。JavaScript 很容易解释这个字符串,而且 JSON 可以表示比"名称/值对"更复杂的结构。例如,可以表示数组和复杂的对象,而不仅仅是键和值的简单列表。

1. 名称/值对

按照最简单的形式,可以用下面的 JSON 表示"名称/值对"。

```
{"firstName":"Brett"}
```

这个示例非常基本,而且实际比等效的纯文本"名称/值对"占用更多的空间。

```
firstName=Brett
```

但当多个"名称/值对"串在一起时,JSON 就体现出它的价值了。首先,可以创建包含多个"名称/值对"的记录,比如:

```
{"firstName":"Brett","lastName":"McLaughlin","email":"aaaa"}
```

从语法方面来看,这与"名称/值对"相比并没有很大的优势,但在有些情况下 JSON 更容易使用,而且可读性更好。例如,它明确地表示以上 3 个值都是同一记录的一部分;花括号使这些值有了某种联系。

2. 数组

当表示一组值时,JSON 不但能够提高可读性,而且可以减少复杂性。假设表示一个人名列表。在 XML 中,需要许多开始标记和结束标记。如果使用典型的名称/值对,必须建立一种专有的数据格式,或者将键名称修改为 person1-firstName 的形式。

如果使用 JSON,只需将多个带花括号的记录分组在一起。

```
{
    "people":[
        {"firstName":"Brett","lastName":"McLaughlin","email":"aaaa"},
        {"firstName":"Jason","lastName":"Hunter","email":"bbbb"},
        {"firstName":"Elliotte","lastName":"Harold","email":"cccc"}
```

```
    ]
}
```

在这个示例中，只有一个名为 people 的变量，值是包含 3 个条目的数组，每个条目是一个人的记录，其中包含名、姓和电子邮件地址。上面的示例说明如何用括号将记录组合成一个值。当然，也可以使用相同的语法表示多个值(每个值包含多个记录)。

```
{
   "programmers": [
      { "firstName": "Brett", "lastName": "McLaughlin", "email": "aaaa" },
      { "firstName": "Jason", "lastName": "Hunter", "email": "bbbb" },
      { "firstName": "Elliotte", "lastName": "Harold","email": "cccc" }
   ],
   "authors": [
      { "firstName": "Isaac", "lastName": "Asimov", "genre": "sciencefiction" },
      { "firstName": "Tad", "lastName": "Williams", "genre": "fantasy" },
      { "firstName": "Frank","lastName": "Peretti", "genre": "christianfiction" }
   ],
   "musicians": [
      { "firstName": "Eric", "lastName": "Clapton", "instrument": "guitar" },
      { "firstName": "Sergei", "lastName": "Rachmaninoff", "instrument":
      "piano" }
   ]
}
```

数组能够表示多个值，每个值进而包含多个值。在不同的主条目(programmers、authors 和 musicians)之间，记录中实际的名称/值对可以不一样。JSON 是完全动态的，允许在 JSON 结构的中间改变表示数据的方式。

处理 JSON 格式的数据时，没有需要遵守的预定义的约束。所以，在同样的数据结构中可以改变表示数据的方式，甚至可以以不同方式表示同一事物。

7.1.3 格式应用

掌握了 JSON 格式，在 JavaScript 中就可以使用它。JSON 是 JavaScript 的原生格式，这意味着在 JavaScript 中处理 JSON 数据不需要任何特殊的 API 或工具包。

1. 赋值给变量

例如，可以创建一个新的 JavaScript 变量，然后将 JSON 格式的数据直接赋值给它。

```
var people={
   "programmers": [
      { "firstName": "Brett", "lastName": "McLaughlin", "email": "aaaa" },
      { "firstName": "Jason", "lastName": "Hunter", "email": "bbbb" },
      { "firstName": "Elliotte", "lastName": "Harold","email": "cccc" }
```

```
    ],
    "authors": [
        { "firstName": "Isaac", "lastName": "Asimov", "genre": "sciencefiction" },
        { "firstName": "Tad", "lastName": "Williams", "genre": "fantasy" },
        { "firstName": "Frank","lastName": "Peretti", "genre": "christianfiction" }
    ],
    "musicians": [
        { "firstName": "Eric", "lastName": "Clapton", "instrument": "guitar" },
        { "firstName": "Sergei", "lastName": "Rachmaninoff", "instrument": "piano" }
    ]
}
```

people 变量包含前面的 JSON 格式的数据，进而可以访问和修改 people 变量。

2. 访问数据

people 变量用点号表示法来表示其中不同的数组元素。如访问 programmers 列表的第一个条目的姓氏，只需在 JavaScript 中使用下面的代码：

```
people.programmers[0].lastName;
```

这行代码首先访问 people 变量中的数据，然后移动到称为 programmers 的条目，再移动到第一个记录([0])，最后访问 lastName 键的值。结果是字符串值 McLaughlin。

下面访问 people 变量的几个不同示例。

```
people.authors[1].genre                 //Value is "fantasy"
people.musicians[3].lastName            //Undefined. This refers to the fourth entry, and
                                          there isn't one
people.programmers[2].firstName         //Value is "Elliotte"
```

利用这样的语法可以处理任何 JSON 格式的数据，而不需要使用任何额外的 JavaScript 工具包或 API。

3. 修改数据

正如可以用点号和方括号访问数据，也可以按照同样的方式轻松地修改数据。

```
people.musicians[1].lastName="Rachmaninov";
```

如果要处理大量 JavaScript 对象，JSON 是一个好选择，这样就可以轻松地将数据转换为可以在请求中发送给服务器端程序的格式。

4. 嵌套格式

许多 JavaScript 树形控件使用 JSON 嵌套格式描述树形结构，如下所示。

```
{
    "id": "100000",
```

```
        "text": "廊坊银行总行",
        "children": [
            {
                "id": "110000",
                "text": "廊坊分行",
                "children": [
                    { "id": "113000", "text": "廊坊银行开发区支行", "leaf": true },
                    { "id": "112000", "text": "廊坊银行解放道支行",
                     "children": [
                         {"id": "112200", "text": "廊坊银行三大街支行", "leaf": true},
                         {"id": "112100", "text": "廊坊银行广阳道支行", "leaf": true }
                       ]
                    },
                    { "id": "111000", "text": "廊坊银行金光道支行", "leaf": true }
                ]
            }
        ]
    }
```

7.1.4 对象方法

JSON 对象不但可以刻画属性，还可以有方法。JSON 对象中的对象与 C++、C# 和 Java 程序设计语言中对象的定义不同，它是直接定义对象，不能通过类的实例化方式定义。而且，JSON 方法与 C++、C# 和 Java 程序设计语言中方法的访问控制方式不同，不能设置其访问权限，没有私有和受保护的方法，是全公开的方法。

1. 方法定义

JSON 方法也是名/值对形式。但名称为函数名，值为一个具体的函数，如下所示。

```
var Rectangle={ length: 50, width:50,  position:{ left: 30, top:30},
                Move : function( x, y ) {
                        Rectangle. position .left=x;
                        Rectangle. position .top=y;
                }
        }
```

上面直接定义了一个矩形对象，含有长、宽、位置 3 个属性和 1 个移动方法。而且位置由左和上 2 个属性刻画。

注意，名/值对中的名可以不用双引号括起来（7. 1 节的名亦如此），值必须是 function(){ }的形式。

2. 方法引用

方法的引用由点号连接对象和其具体方法组成，例如：

```
Rectangle.Move(100, 100);
```

上面的语句表示将矩形左上角移动到(100，100)处。

7.2 上机实验样例

1. 实验目标

掌握综合运用 JSON 和 jQuery 进行面向对象的 JavaScript 程序设计方法。

2. 程序功能

浏览器输出一个矩形，周而复始地从左向右、从右向左水平运动。矩形用一个 div 表示。矩形的运动状态用对象来刻画。

3. 实验步骤

(1) 源程序：7-2.htm

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
        <script src="Scripts/jquery-1.4.1.js" type="text/javascript">
        </script>
        <script type="text/javascript">
            var Rectangle={
                Position: { X: 10, Y: 10 }, Step: 10, Direction: 1,
                Set: function () {
                    $("#div1").css("left", Rectangle.Position.X+"px");
                    $("#div1").css("top", Rectangle.Position.Y+"px");
                },
                Move: function () {
                    if (Rectangle.Position.X==510) Rectangle.Direction=-1;
                    else if (Rectangle.Position.X==10) Rectangle.Direction=1;
                    Rectangle.Position.X+=Rectangle.Direction * Rectangle.Step;
                    Rectangle.Set();
                },
                Play: {
                    Timer: null,
                    Start: function () {
                        Rectangle.Set ();
                        if (Rectangle.Play.Timer==null)
                            Rectangle.Play.Timer=setInterval(Rectangle.Move,
                            100);
```

```
                    },
                    End: function () {
                         clearInterval(Rectangle.Play.Timer); Rectangle.Play.Timer=
                         null;
                    }
               }
          };
     </script>
</head>
<body>
     <div>
          <input type="button" onclick="Rectangle.Play.Start();" value="启动"/>
          <input type="button" onclick="Rectangle.Play.End();" value="停止"/>
     </div>
     <div id="div1" style="background-color:#bd92a3; position:absolute; width:
     250px; height:250px;"></div>
     <script type="text/javascript">
          Rectangle.Play.Start();
     </script>
</body>
</html>
```

(2) 程序分析

Rectangle 为一个对象，用于控制一个 div 元素的水平运动。该对象有 3 个属性、2 个方法、1 个子对象。

① 3 个属性。

Rectangle. Position：含有 2 个分量，即 X 和 Y，存储矩形左上角坐标。

Rectangle. Step：步长，矩形运动一次的水平增量。

Rectangle. Direction：运动方向，取值有 1 和 −1，分别表示从左向右和从右向左运动。

② 2 个方法。

Rectangle. Set()：将 Rectangle. Position 的分量 X 和 Y 作为 div 元素的 left 和 top 属性值。

Rectangle. Move()：根据 Rectangle. Position 坐标控制 div 元素运动一次。如果当前矩形对象水平坐标为 510，将运动方向设为从右向左，即 1。如果当前矩形水平坐标为 10，将运动方向设为从左向右，即 −1。

③ 1 个子对象。

Play 子对象具有 1 个属性和 2 个方法，即 Rectangle. Play. Timer 属性、Rectangle. Play. Start() 和 Rectangle. Play. End()。

Rectangle. Play. Timer：存储定时器，初始值为空。

Rectangle. Play. Start()：首先执行 Rectangle. Set()。而后如果 Rectangle. Play.

Timer 为空，则用 setInterval()方法创建定时器。setInterval()的定时间隔为 100ms，所执行的方法为 Rectangle. Move()。

(3) 现象分析

如果去掉 Rectangle. Play. Start() 函数内的语句 if(Rectangle. Play. Timer==null)，可把 Rectangle. Play. Start()变成：

```
Start: function () {
    Rectangle.Set ();
    if (Rectangle.Play.Timer==null) Rectangle.Play.Timer=setInterval
    (Rectangle.Move, 100);
}
```

① 越点启动速度越快。

未单击停止按钮，越点启动按钮，矩形运动越快。这是因为去掉语句 if(Rectangle. Play. Timer==null)后，每点一次启动按钮，无论定时器是否已创建，都执行 setInterval()方法创建定时器 1 次。这样，单击多少次启动按钮，就会有多少个定时器。而这些定时器执行同一个 Rectangle. Move()方法，所以越来越快。

② 单击停止按钮无效。

多次单击启动按钮后，无论怎么单击停止按钮，都无法使矩形的运动停下来。这是因为去掉语句 if(Rectangle. Play. Timer==null)后，每单击一次启动按钮，无论定时器是否已创建，都执行 setInterval()方法创建定时器 1 次。而且，Rectangle. Play. Timer 属性变量只存储最后创建的定时器，之前创建的定时器未被存储，因而停不下来。

7.3 实验任务

1. 实验题目

运用 JSON 进行面向对象动画脚本程序设计。

2. 程序功能

设计 1 个动画网页，至少含有 1 个 div 元素和 4 个按钮。用 div 元素展现正方形，4 个按钮分别用于启动、停止、1 倍速和 2 倍速按钮。

启动网页，正方形周期性地匀速从左上角运动到右下角，从右下角运动到左上角。单击停止按钮，则运动停止。单击启动按钮，则从当前位置按预定方向运动。

3. 实验目的

(1) 掌握运用 JSON 进行面向对象的脚本程序开发方法。

(2) 深化动画机制的实现。

4. 实验类型

综合设计。

5. 实验要求

(1) 脚本程序实现完全运用 jQuery 程序实现。
(3) 完全运用 JSON 进行面向对象的程序开发。
(2) 倍速只能用修改步长的方式实现,不能用修改频率的方式实现。

6. 实验环境

(1) 计算机:PC、内存 8GB、主频 1.8GHz 及以上、硬盘 500GB 及以上。
(2) 操作系统:Windows XP、Windows 7、Windows 8、Windows 10。
(3) 开发环境:Visual Studio 2010。
(4) 浏览器:IE9 及以上、Chrome、Firefox、Safari、Edge、QQ 浏览器等。

7. 实验原理

给出与事件、CSS 添加删除元素直接相关的 jQuery 语句,并辅以必要的说明与分析。

8. 遇到的问题及解决办法

(1) 给出所遇到的全部错误现象描述。
(2) 给出修正错误前设定的断点位置。
(3) 给出修正错误所监视的变量。
(4) 给出运行到断点处监视变量值。

9. 运行结果

给出所完成任务功能的屏幕截图。

第 8 章



HTML5

■ **知识目标**

- 理解 HTML5 的新特性、优势和不足
- 掌握 HTML5 画布和 SVG 绘图技术
- 掌握 HTML5 Web 存储、SSE、Web Worker 和 Web 缓存技术
- 理解 Web Socket 的通信机制

■ **能力目标**

- 根据实际需求选用恰当的 HTML5 元素
- 根据实际情况运用画布进行绘图
- 结合 HTML5 新元素，综合运用 jQuery 事件、jQuery CSS、JSON 开发 Web 网页

■ **素质目标**

- 应用 HTML5 开发用户体验良好的网页

■ **教学重点**

- 运用 JSON 封装画布绘图功能和定时器实现动画功能

■ **教学难点**

- 绘图与动画算法的设计与实现

■ **建议学时**

- 理论：6 学时
- 实验：6 学时

8.1 HTML5 概述

HTML5 草案的前身为 Web Applications 1.0，2004 年由 WHATWG 提出，2007 年被 W3C 接纳，并成立了新的 HTML 工作团队。HTML5 的第一份正式草案已于 2008 年 1 月 22 日公布但仍处于完善之中。然而，大部分现代浏览器已经具备了某些 HTML5 支持特征。2012 年 12 月 17 日，万维网联盟(W3C)宣布凝结了大量网络工作者心血的 HTML5 规范已经正式定稿。W3C 称 HTML5 是开放的 Web 网络平台的奠基石。

2013年5月6日，HTML 5.1正式草案公布。该规范定义了第5次重大版本，第1次要修订万维网的核心语言：超文本标记语言（HTML）。在这个版本中，新功能不断推出，以帮助Web应用程序员努力提高新元素互操作性。

支持HTML5的浏览器包括Firefox、IE 9及其更高版本、Chrome、Safari、Opera等，国内的傲游浏览器（Maxthon）、基于IE或Chromium（Chrome的工程版或称实验版）所推出的360浏览器、搜狗浏览器、QQ浏览器、猎豹浏览器等国产浏览器同样具备支持HTML5的能力。

在移动设备上开发HTML5应用有两种方法：使用HTML5的语法和使用JavaScript引擎。JavaScript引擎的构建方法让制作手机网页游戏成为可能。由于界面层很复杂，可选用公开的功能丰富强大的UI工具包开发界面。

HTML5手机应用的最大优势是可以在网页上直接调试和修改。原先应用开发人员可能需要花费大力气才能达到HTML5的同等效果，而且不断地重复编码、调试和运行，这需要改善。因此，许多手机杂志客户端基于HTML 5标准，开发人员可以轻松地调试修改。

2014年10月29日，万维网联盟宣布，经过近乎8年的艰辛努力，HTML5标准规范终于制定完成了，并已公开发布。

HTML5将取代1999年制定的HTML 4.01、XHTML 1.0标准，以期在互联网应用迅速发展的时候，使网络标准符合当代的网络需求，为桌面和移动平台带来无缝衔接的丰富内容。

1. 新特性

HTML 5的设计目的是在移动设备上支持多媒体。新语法特征被引进，以支持设计目的，如video、audio和canvas标记。HTML5开发的新功能可以真正改变用户与文档的交互方式，包括：

① 新元素：用于绘画的canvas元素、用于媒介回放的video和audio元素、article、footer、header、nav、section等。

② 新表单控件：如calendar、date、time、email、url、search。

③ 离线缓存。

④ 新属性。

⑤ 完全支持CSS3。

⑥ 本地数据。

⑦ Web应用：Web Socket、SSE、Web SQL。

⑧ 拖放功能。

⑨ 取消过时的HTML 4标签。

2. 应用革新

（1）带来一个无缝网络

HTML5会带来一个统一的网络，无论是笔记本、台式机，还是智能手机，都可以很

方便地浏览基于 HTML5 的网站。因此,设计网站时,开发者需要重新考虑用户体验、网站浏览、网站结构等因素,使网站在任何硬件设备上都通用。

(2) 变成企业的 SAAS 平台

一些知名企业,如微软、Sales force、SAP Sybase 正在开发 HTML5 的开发工具。如果正在构建企业应用,选择 HTML5 是公司部署 SAAS 战略的首选。

(3) 变得很移动

目前,很多人热衷于开发独立的移动应用,但 HTML5 很可能会是独立移动应用的终结者。由于 HTML5 将应用的功能直接加入其内核,因此很可能引导移动技术潮流重新回到浏览器时代。HTML5 允许开发者在浏览器内开发应用,所以制定一项桌面或者移动应用的长期发展策略的设计者需要考虑这一点。

3. 优势

(1) 网络标准

HTML5 本身是由 W3C 推荐出来的,是通过谷歌、苹果、诺基亚、中国移动等几百家公司一起酝酿开发的技术,最大的好处在于公开性。换句话说,每一个公开的标准都可以根据 W3C 的资料库找寻根源。另一方面,W3C 通过的 HTML5 标准也意味着每一个浏览器或每一个平台都会去实现。

(2) 多设备跨平台

HTML5 的优点,主要在于可以进行跨平台使用。比如开发了一款 HTML5 的游戏,可以很轻易地移植到 UC 的开放平台、Opera 的游戏中心、Facebook 的应用平台,甚至可以通过封装技术发放到 App Store 或 Google Play 上。所以它的跨平台性非常强大,这也是大多数人对 HTML5 有兴趣的主要原因。

(3) 自适应网页设计

一次设计,普遍适用,让同一张网页自动适应不同大小的屏幕,根据屏幕宽度自动调整布局(layout)。这解决了传统网页的困境:为不同的设备提供不同的网页,比如专门提供一个 mobile 版本,或者 iPhone / iPad 版本,导致同时维护多个版本,而如果网站有多个 portal(入口),则会极大增加架构设计的复杂度。

(4) 即时更新

C/S 结构的游戏客户端每次都要更新,成本较高。而更新 HTML5 游戏就好像更新页面一样,是即时更新。

综上所述,HTML5 有以下优点。

① 提高可用性和改进用户的友好体验。

② 有几个新的标签,有助于开发人员定义重要的内容。

③ 可以给站点带来更多的多媒体元素(视频和音频)。

④ 可以很好地替代 Flash 和 Silverlight。

⑤ 涉及网站的抓取和索引时,对 SEO 很友好。

⑥ 将被大量应用于移动应用程序和游戏。

⑦ 可移植性好。

HTML5 亦有不足。首先,该标准并未很好地被浏览器所支持。其次,因新标签的引入,各浏览器之间将缺少一种统一的数据描述格式,造成用户体验不佳。

8.2 HTML5 基础

一个含有视频、音频的简单 HTML 文档如下。

(1) 例 8-2-1-1.htm 源文档

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
        <title></title>
    </head>
    <body>
        <video width="320" height="240" controls="controls" autoplay="autoplay" src="
        AudioVideo/movie.mp4">
          你的浏览器不支持 video 标签。
        </video>
        <audio src="AudioVideo/周杰伦-双节棍.mp3" controls="controls" />
    </body>
</html>
```

(2) VS2010 解决方案视图

VS2010 解决方案视图如图 8.1 所示。

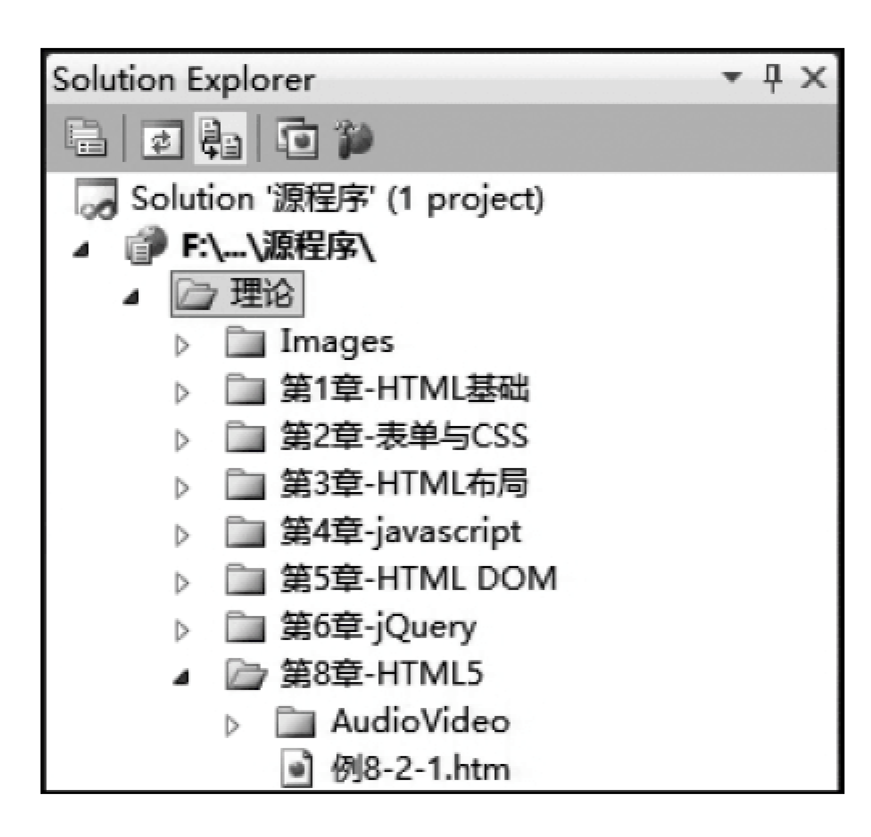


图 8.1　VS2010 解决方案视图

(3) 浏览器显示结果

浏览器显示结果如图 8.2 所示。

(4) 文档解析

<! doctype> 声明必须位于 HTML5 文档中的第 1 行,用 html 表示 HTML5 版本,如下所示。



图 8.2 浏览器显示结果

```
<!DOCTYPE html>
```

对于中文网页，需要使用<meta charset="utf-8">声明编码，否则会出现乱码。

文档 body 含有 2 个 HTML 新元素，依次为视频元素和音频元素。

8.2.1 canvas

<canvas>标签定义图形比，如图、表、文字和其他图像等。它只是图形容器，需要使用脚本来绘制图形。可以通过多种脚本方法绘制路径、盒、圆、字符，以及添加图像等。

1. 浏览器支持

支持 canvas 标签的各浏览器最低版本依次为：Chrome 4.0、IE 9.0、Friefox 2.0、Safari 3.1、Opera 9.0。

2. 创建一个画布

默认情况下，<canvas> 元素没有边框和内容。如果让<canvas> 元素具有边框，需设置 style 的 boder 属性。

<canvas>没有边框的简单实例如下。

```
<canvas id="myCanvas" width="200" height="100"></canvas>
```

标签通常需要指定一个 id 属性，以便在脚本中引用，width 和 height 属性定义画布的宽和高。而且可以在 HTML 页面中使用多个<canvas>元素。

使用 style 属性来添加边框的实例如下。

```
<canvas id="myCanvas" width="200" height="100" style="border:1px solid #000000;">
</canvas>
```

3. 使用 JavaScript 绘制图像

canvas 元素本身是没有绘图能力的。所有的绘制工作在 JavaScript 内部完成，如例 8-2-1-1 所示。

例 8-2-1-1 画布绘图的简单示例。

(1) 例 8-2-1-1. htm 源文档

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
        <title></title>
        <script src="../第 6 章-jQuery/Scripts/jquery-1.4.1.min.js" type="text/
        javascript"></script>
        <script type="text/javascript">
            $(document).ready(function () {
                var c=document.getElementById("myCanvas");
                var ctx=c.getContext("2d");
                ctx.fillStyle="#FF0000";
                ctx.fillRect(10, 10, 150, 75);
            }
            );
        </script>
    </head>
    <body>
        <canvas id="myCanvas" width="200" height="100" style="border:1px dashed #
       000000;">
          你的浏览器不支持 canvas 标签。
        </canvas>
    </body>
</html>
```

(2) 浏览器显示结果

浏览器显示结果如图 8.3 所示。

图 8.3 例 8-2-1-1 的浏览器显示结果

(3) 实例解析

首先找到<canvas>元素：

```
var c=document.getElementById("myCanvas");
```

然后创建 context 对象：

```
var ctx=c.getContext("2d");
```

getContext("2d")对象是内建的 HTML5 对象，拥有多种绘制路径、矩形、圆形、字

符,以及添加图像的方法。

最后,以下两行代码绘制一个红色的矩形:

```
ctx.fillStyle="#FF0000";
ctx.fillRect(0,0,150,75);
```

设置 fillStyle 属性可以是 CSS 颜色、渐变或图案。fillStyle 默认设置是#000000(黑色)。fillRect(x,y,width,height) 方法定义了矩形当前的填充方式。

4. canvas 坐标

canvas 是一个二维网格,左上角坐标为 (0,0),画布内任何元素坐标均相对于左上角。画布二维坐标系如图 8.4 所示。

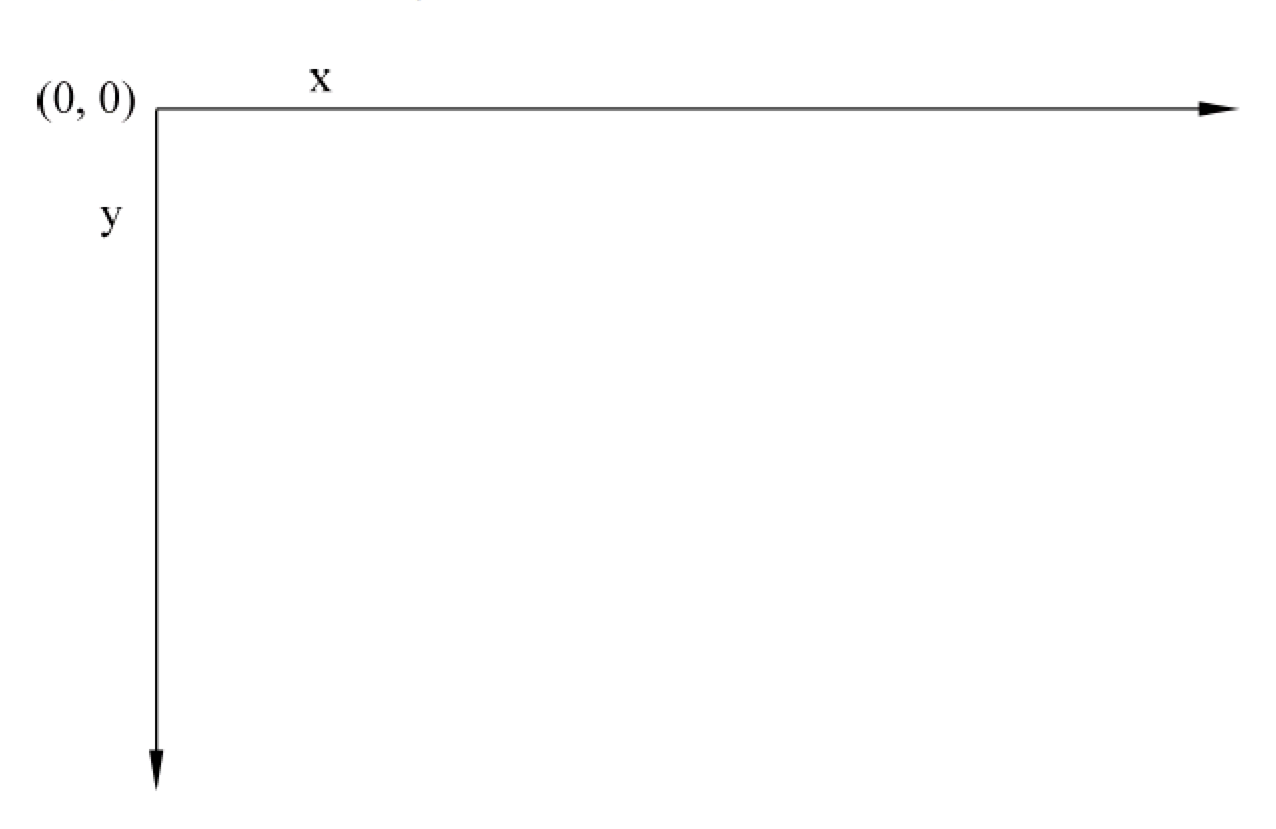


图 8.4　画布二维坐标系

例 8-2-1-1 的 fillRect 方法拥有参数 (0,0,150,75),说明在画布上左上角 (0,0)开始绘制 150×75 的矩形。

5. 画线

在 canvas 上画线,需要使用以下 2 种方法。

① moveTo(x,y) 定义线条开始坐标。

② lineTo(x,y) 定义线条结束坐标。

绘制线条用到 ink 的方法,就像 stroke(),如例 8-2-1-2 所示。

例 8-2-1-2　使用 stroke()方法,在坐标(0,0)和坐标(200,100)之间画一条直线。

JavaScript 内部程序如下。

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.moveTo(0,0);
ctx.lineTo(200,100);
ctx.stroke();
```

浏览器显示结果如图 8.5 所示。

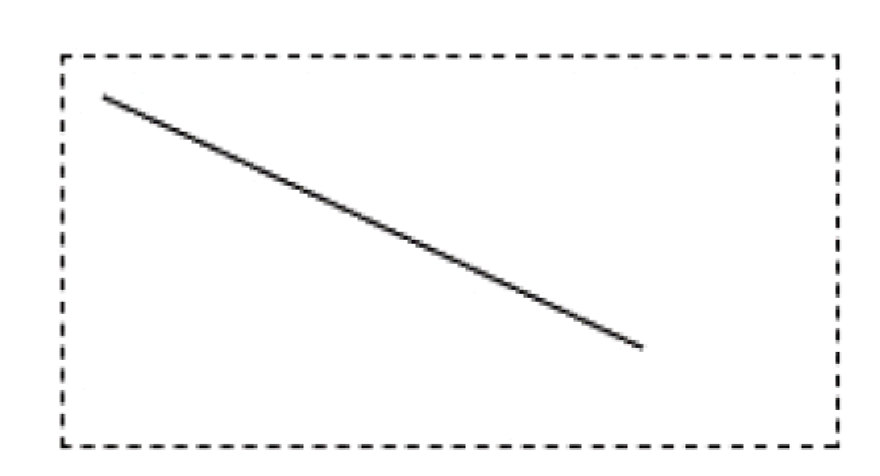


图 8.5　例 8-2-1-2 的浏览器显示结果

6. 画圆

在 canvas 中绘制圆形，方法为：arc(x，y，r，start，stop)。

实际上，在绘制圆形时还需使用 ink 的方法，比如 stroke()或者 fill()。

例 8-2-1-3 使用 arc()方法绘制一个圆。

JavaScript 内部程序如下。

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.beginPath();
ctx.arc(95, 50, 40, 0, 2*Math.PI);
ctx.stroke();
```

浏览器显示结果如图 8.6 所示。

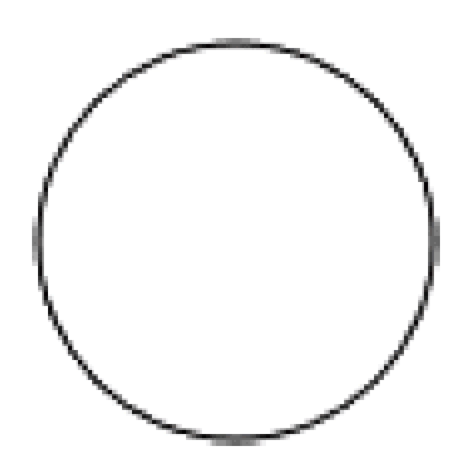


图 8.6 例 8-2-1-3 的浏览器显示结果

7. 文本

使用 canvas 绘制文本，需要的重要属性和方法如下。

① font：定义字体。

② fillText(text，x，y)：在 canvas 上绘制实心的文本。

③ strokeText(text，x，y)：在 canvas 上绘制空心的文本。

(1) 使用 fillText()

例 8-2-1-4 使用 Arial 字体在画布上绘制一个高 30px 的实心文字。

① JavaScript 内部程序。

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.font="30px Arial";
ctx.fillText("Hello World",10,50);
```

② 浏览器显示结果。

浏览器显示结果如图 8.7 所示。

Hello World

图 8.7 实心文字

(2) 使用 strokeText()

例 8-2-1-5 使用 Arial 字体在画布上绘制一个高 30px 的空心文字。

① JavaScript 内部程序。

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
```

```
ctx.font="30px Arial";
ctx.strokeText("Hello World",10,50);
```

② 浏览器显示结果。

浏览器显示结果如图 8.8 所示。

Hello World

图 8.8 空心文字

8. 渐变

渐变可以填充在矩形、圆形、线条、文本中，各种形状可以自己定义不同的颜色。

有以下两种不同的方式来设置 Canvas 渐变。

① createLinearGradient(x,y,x1,y1)：创建线条渐变。

② createRadialGradient(x,y,r,x1,y1,r1)：创建一个径向/圆渐变。

使用渐变对象，必须使用两种或两种以上的停止颜色。addColorStop()方法指定颜色停止，使用 0～1 之间的参数值来描述。使用渐变，设置 fillStyle 或 strokeStyle 的值为渐变对象，然后绘制形状，如矩形、文本或一条线。

(1) 使用 createLinearGradient()

例 8-2-1-6 创建一个线性渐变，用其填充矩形。

① JavaScript 内部程序。

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
var grd=ctx.createLinearGradient(0,0,200,0);
grd.addColorStop(0,"red");
grd.addColorStop(1,"white");
ctx.fillStyle=grd;
ctx.fillRect(10,10,150,80);
```

② 浏览器显示结果。

浏览器显示结果如图 8.9 所示。



图 8.9 线性渐变

(2) 使用 createRadialGradient()

例 8-2-1-7 创建一个径向/圆渐变，用其填充矩形。

① JavaScript 内部程序。

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
var grd=ctx.createRadialGradient(75,50,5,90,60,100);
grd.addColorStop(0,"red");
grd.addColorStop(1,"white");
ctx.fillStyle=grd;
ctx.fillRect(10,10,150,80);
```

② 浏览器显示结果。

浏览器显示结果如图 8.10 所示。

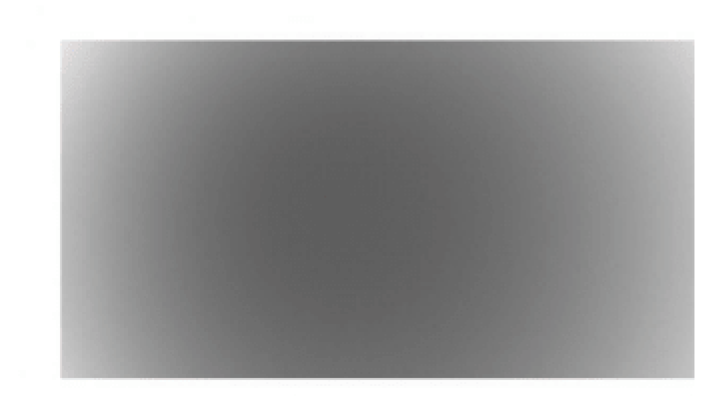


图 8.10 径向渐变

9. 图像

把一幅图像放置到画布上，方法为：drawImage(image，x，y)。

例 8-2-1-8 把一幅图像放置到画布上。

(1) 例 8-2-1-8.htm 源文档

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
        <title></title>
          <script src="../第 6 章 - jQuery/Scripts/jquery - 1.4.1.min.js" type =
          "text/javascript"></script>
    </head>
    <body>
        <img id="scream" src="../images/img_the_scream.jpg" alt="The Scream" width="
        220" height="277"/>
        <canvas id="myCanvas" width="240" height="297" style="border:1px
        dashed #CCCCCC">
          你的浏览器不支持 canvas 标签。
        </canvas>
        <script type="text/javascript">
            $("img").hide();
            var c=document.getElementById("myCanvas");
            var ctx=c.getContext("2d");
            var img=document.getElementById("scream");
            ctx.drawImage(img, 10, 10);
        </script>
    </body>
</html>
```

(2) 浏览器显示结果

浏览器显示结果如图 8.11 所示。



图 8.11 例 8-2-1-8 的浏览器显示结果

8.2.2 内联 SVG

SVG 指可伸缩矢量图形（Scalable Vector Graphics），遵从万维网联盟的标准，用来定义用于网络的基于矢量的图形，使用 XML 格式定义图形。在放大或改变尺寸的情况下，SVG 图像不失真。

1. SVG 优势

与其他图像格式相比（如 JPEG 和 GIF），使用 SVG 的优势如下。

① SVG 图像可通过文本编辑器来创建和修改。

② SVG 图像可被搜索、索引、脚本化或压缩。

③ SVG 图像可伸缩。

④ SVG 图像可在任何分辨率下被高质量地打印。

⑤ SVG 可在图像质量不下降的情况下被放大。

2. 浏览器支持

IE 9+、Firefox、Opera、Google、Chrome 和 Safari 支持内联 SVG。

3. 把 SVG 直接嵌入 HTML 页面

HTML5 能将 SVG 元素直接嵌入 HTML 页面中，如例 8-2-2-1 所示。

例 8-2-2-1 用 SVG 在浏览器中画一个五角星。

（1）例 8-2-2-1.htm 源文档

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
        <title></title>
    </head>
    <body>
        <svg xmlns="http://www.w3.org/2000/svg" version="1.1" height="190">
          <polygon points="100,10 40,180 190,60 10,60 160,180"
          style="fill:lime;stroke:purple;stroke-width:5;fill-
          rule:evenodd;">
        </svg>
    </body>
</html>
```

(2) 浏览器显示结果

浏览器显示结果如图 8.12 所示。

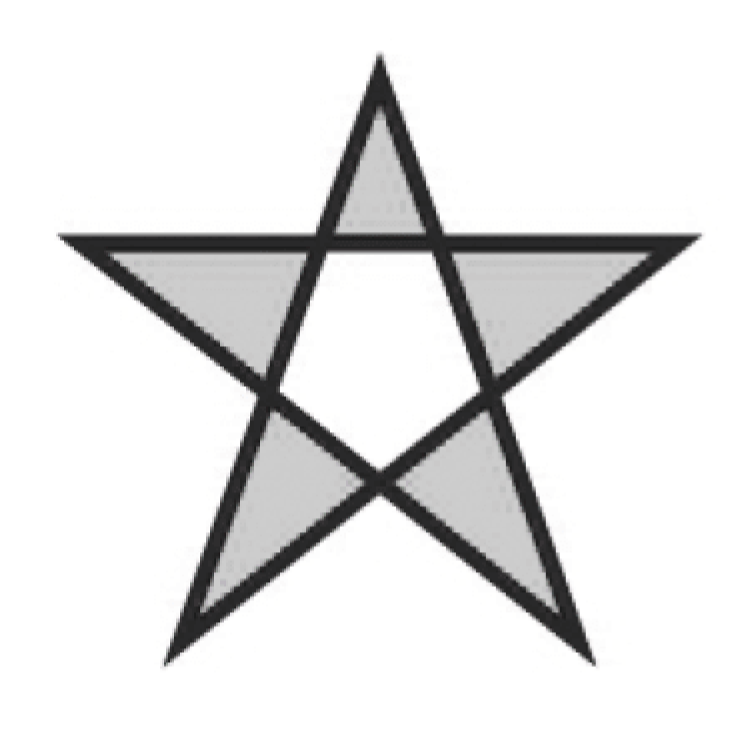


图 8.12　例 8-2-2-1 的浏览器显示结果

4. SVG 与 canvas 的区别

SVG 是一种使用 XML 描述 2D 图形的语言。canvas 通过 JavaScript 来绘制图形。SVG 基于 XML,这意味着 SVG DOM 中的每个元素都是可用的,可以为某个 SVG 元素加 JavaScript 事件处理器。在 SVG 中,每个被绘制的图形均被视为对象。如果 SVG 对象的属性发生变化,那么浏览器能够自动重现图形。canvas 是逐像素进行渲染的。在 canvas 中,一旦图形被绘制完成,就不会继续得到浏览器的关注。如果其位置发生变化,那么整个场景也需要重新绘制,包括任何或许已被图形覆盖的对象。表 8.1 列出了 canvas 与 SVG 的特点比较。

表 8.1　canvas 与 SVG 的特点比较

canvas	SVG
依赖分辨率	不依赖分辨率
不支持事件处理器	支持事件处理器
弱的文本渲染能力	最适合带有大型渲染区域的应用程序(如谷歌地图)
能够以.png 或.jpg 格式保存结果图像	复杂度高会减慢渲染速度(任何过度使用 DOM 的应用都不快)
最适合图像密集型的游戏,其中的许多对象会被频繁重绘	不适合游戏应用

8.2.3 Web 存储

使用 HTML5 可以在本地存储用户的浏览数据。HTML5 Web 存储是一个比 cookie 更好的本地存储方式。先前版本的 HTML 本地存储使用 cookie。但是 HTML5 Web 存储需要更加安全与快速。这些数据不会被保存在服务器上,但是这些数据只用于用户请求网站数据上。它也可以存储大量的数据,而不影响网站的性能。Web 存储数据以键/值对存在,Web 网页的数据只允许该网页访问使用。

1. 浏览器支持

IE 8+、Firefox、Opera、Chrome 和 Safari 支持 Web 存储。

2. localStorage 和 sessionStorage

客户端存储数据的两个对象如下。

① localStorage:没有时间限制的数据存储。

② sessionStorage：针对一个 session 的数据存储。

使用 Web 存储前，应检查浏览器是否支持 localStorage 和 sessionStorage，运用如下语句判断。

```
if(typeof(Storage)!=="undefined")
{
    //支持相关的一些代码.....
}
else
{
    //抱歉!不支持 web 存储
}
```

3. localStorage 对象

localStorage 对象存储的数据没有时间限制，例如。

```
localStorage.sitename="HTML5 教程";
document.getElementById("result").innerHTML="网站名: "+localStorage.sitename;
```

使用 key＝"sitename"和 value＝" HTML5 教程"可以创建一个 localStorage 键/值对。检索键名为"sitename"的值，然后将数据插入 id＝"result"的元素中。

移除 localStorage 中的"sitename "键，语句如下：

```
localStorage.removeItem("lastname");
```

localStorage 和 sessionStorage 都使用相同的 API，常用的 API 如下(以 localStorage 为例)。

① 保存数据：localStorage. setItem(key，value)。

② 读取数据：localStorage. getItem(key)。

③ 删除单个数据：localStorage. removeItem(key)。

④ 删除所有数据：localStorage. clear()。

⑤ 得到某个索引的 key：localStorage. key(index)。

键/值对通常以字符串存储，可根据需要转换该格式，如例 8-2-3-1 所示。

例 8-2-3-1 展示了用户单击按钮的次数。

(1) 例 8-2-3-1. htm 源文档

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
          <script src="../第 6 章 - jQuery/Scripts/jquery - 1. 4. 1. min. js" type =
          "text/javascript"></script>
        <title></title>
        <script>
```

```
            function clickCounter() {
                if (typeof (Storage) !=="undefined") {
                    if (localStorage.clickcount) {
                        localStorage.clickcount=Number(localStorage.
                        clickcount)+1;
                    }
                    else {
                        localStorage.clickcount=1;
                    }
                    $("#result").html("已单击按钮"+localStorage.clickcount+"次");
                }
                else {
                    $("#result").html("对不起,您的浏览器不支持 web 存储。");
                }
            }
        </script>
    </head>
    <body>
        <p><button onclick="clickCounter()" type="button">点我!</button>
        </p>
        <div id="result"></div>
        <p>单击该按钮查看计数器的增加。</p>
        <p>关闭浏览器选项卡(或窗口),重新打开此页面,计数器将继续计数(不是重置)。
        </p>
    </body>
</html>
```

(2) 浏览器显示结果

浏览器显示结果如图 8.13 所示。

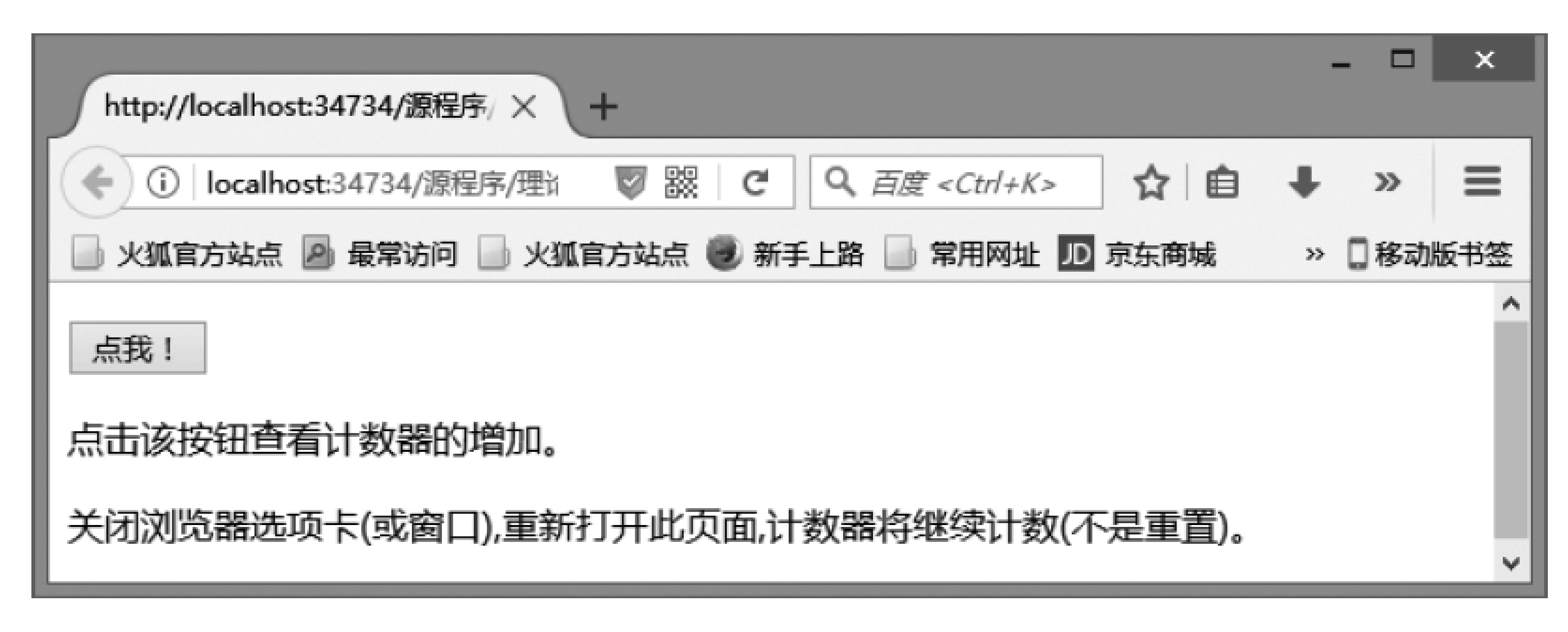


图 8.13　例 8-2-3-1 的浏览器显示结果

4. sessionStorage 对象

sessionStorage 方法针对一个 session 进行数据存储。关闭浏览器窗口后,数据会被

删除。

8.2.4 应用程序缓存

HTML5 引入了应用程序缓存,这意味着 Web 应用可进行缓存,并可在没有因特网连接时访问。使用 HTML5 应用程序缓存,通过创建 Cache Manifest 文件可以创建 Web 应用的离线版本。应用程序缓存为应用带来以下 3 个优势。

① 离线浏览:用户可在应用离线时使用它们。

② 速度:已缓存资源加载得更快。

③ 减少服务器负载:浏览器将只从服务器下载更新过或更改过的资源。

1. 浏览器支持

IE 10.0 以上版本、Firefox、Chrome、Safari 和 Opera 支持应用程序缓存。

2. Cache Manifest 示例

下面的例子展示了带有 Cache Manifest 的 HTML 文档(供离线浏览)。

```
<!DOCTYPE HTML>
<html manifest="demo.appcache">
    <body>
        文档内容……
    </body>
</html>
```

3. Cache Manifest 基础

启用应用程序缓存,需在文档的<html>标签中包含 manifest 属性。

```
<!DOCTYPE HTML>
<html manifest="demo.appcache">
...
</html>
```

每个指定了 manifest 的页面在用户访问时都会被缓存。如果未指定 manifest 属性,则页面不会被缓存(除非在 manifest 文件中直接指定了该页面)。

Manifest 文件建议的文件扩展名是.appcache。

Manifest 文件需要配置正确的 MIME-type,即 text/cache-manifest。必须在 Web 服务器上进行配置。

4. Manifest 文件

Manifest 文件是简单的文本文件,它告知浏览器被缓存的内容(以及不缓存的内容)。Manifest 文件可分为以下 3 个部分。

① Cache Manifest:在此标题下列出的文件将在首次下载后缓存,必须有标题。

② Network：在此标题下列出的文件需要与服务器连接，且不会被缓存。

③ Fallback：在此标题下列出的文件规定页面无法访问时的回退页面（如 404 页面）。

（1）Cache Manifest

Cache Manifest 为 Manifest 的第 1 行，是必需的，如下所示。

```
Cache Manifest
/theme.css
/logo.gif
/main.js
```

上面的 manifest 文件列出了需要缓存的文件对象：一个 CSS 文件、一个 GIF 图像及一个 JavaScript 文件。当加载 manifest 文件后，浏览器会从网站的根目录下载这 3 个文件。然后，无论用户何时与因特网断开连接，这 3 个文件依然可用。

Network

（2）下面的 Network 小节规定文件 login. php 不被缓存，离线时不可用。

```
Network:
login.php
```

可以使用星号来指示所有其他其他资源/文件都需要因特网连接。

```
Network:
 *
```

（3）Fallback

下面的 Fallback 小节规定如果无法建立因特网连接，则用 offline. html 替代/html5/目录中的所有文件。

```
Fallback:
/html/ /offline.html
```

5. 更新缓存

一旦应用被缓存，它就会保持缓存，直到发生下列情况。

① 用户清空浏览器缓存。

② manifest 文件被修改（参阅提示）。

③ 由程序来更新应用缓存。

6. 完整的 Manifest 文件

```
Cache Manifest
# 2012-02-21 v1.0.0
/theme.css
/logo.gif
/main.js
```

```
Network:
login.php
Fallback:
/html/ /offline.html
```

以"#"开头的是注释行,但也可满足其他用途。应用的缓存会在其 manifest 文件更改时被更新。如果编辑了一幅图片,或者修改了一个 JavaScript 函数,这些改变都不会被重新缓存。更新注释行中的日期和版本号是一种使浏览器重新缓存文件的方法。

一旦文件被缓存,浏览器会继续展示已缓存的版本,即使修改了服务器上的文件。为了确保浏览器更新缓存,需要更新 manifest 文件的注释行日期等。

8.2.5 Web Workers

当在 HTML 页面中执行脚本时,页面状态是不可响应的,直到脚本已完成。Web Worker 是运行在后台的 JavaScript,独立于其他脚本,不会影响页面的性能。可以继续做任何愿意做的事情:单击、选取内容等,而此时 Web Worker 在后台运行。

1. 浏览器支持

IE 10.0、Firefox、Chrome、Safari 和 Opera 都支持 Web Workers。

2. 一个 Web Workers 实例

例 8-2-5-1 页面显示计数值,用一个简单的 Web Worker 在后台进行定时计数。

(1) 前台网页:例 8-2-5-1.htm 源文档

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
        <title></title>
    </head>
    <body>
        <p>计数: <output id="result"></output></p>
        <button onclick="startWorker()">开始工作</button>
        <button onclick="stopWorker()">停止工作</button>
        <p>
            <strong>注意:</strong>
            Internet Explorer 9 及更早 IE 版本浏览器不支持 Web Workers.
          </p>
        <script>
            var w=null;
            function startWorker() {
                if (typeof (Worker) !=="undefined") {
                    if (w==null) {
```

```
                    w=new Worker("例 8-2-5-1.js");
                }
                w.onmessage=function (event) {
                    document.getElementById("result").innerHTML=event.data;
                };
            }
            else {
                document.getElementById("result").innerHTML="浏览器不支持 Web
                Workers.";
            }
        }
        function stopWorker() {
            w.terminate();
            w=null;
        }
    </script>
  </body>
</html>
```

(2) 后台 javaScript 程序：例 8-2-5-1.js 文件代码

```
var i=0;
function timedCount()
{
    i=i+1;
    postMessage(i);
    setTimeout("timedCount()",500);
}
timedCount();
```

(3) 浏览器显示结果

浏览器显示结果如图 8.14 所示。



图 8.14 例 8-2-5-1 的浏览器显示结果

(4) 实例解析

① 检测浏览器是否支持。

创建 Web Worker 之前，运用如下语句检测用户浏览器是否支持它。

```
if(typeof(Worker)!=="undefined")
{
    //是的！Web Worker 支持！
    //一些代码.....
}
else
{
    //抱歉！Web Worker 不支持
}
```

② 创建 Web Worker 文件。

在一个外部 JavaScript 文件(例 8-2-5-1.js)中创建 Web Worker 需要执行的计数脚本程序。

```
var i=0;
function timedCount()
{
    i=i+1;
    postMessage(i);
    setTimeout("timedCount()",500);
}
timedCount();
```

以上代码中重要的部分是 postMessage()方法，它用于向 HTML 页面传回一段消息。

Web Worker 通常不用于如此简单的脚本程序，而是用于更耗费 CPU 资源的任务。

③ 创建 Web Worker 对象。

有了 Web Worker 文件后，需要从 HTML 页面调用它。下面的代码检测是否存在 Worker，如果不存在，则创建一个新的 Web Worker 对象，然后运行例 8-2-5-1.js 中的代码。

```
if(typeof(w)=="undefined")
{
    w=new Worker("demo_workers.js");
}
```

④ 发送和接收消息。

然后可从 Web Worker 发送和接收消息。此时需向 Web Worker 添加一个 onmessage 事件监听器，语句如下：

```
w.onmessage=function(event){
    document.getElementById("result").innerHTML=event.data;
};
```

⑤ 终止 Web Worker。

创建 Web Worker 对象后，它会继续监听消息（即使在外部脚本完成之后），直到其被终止为止。如需终止 Web Worker，并释放浏览器/计算机资源，则使用 terminate()方法，如下所示：

```
w.terminate();
```

8.2.6 服务器发送事件(Server-Sent Events)

HTML5 服务器发送事件（Werver-Went Event）允许网页获得来自服务器的更新。Server-Sent 事件指的是网页自动获取来自服务器的更新。以前也可能做到这一点，方法是网页不得不询问是否有可用的更新。通过服务器发送事件，更新能够自动到达。服务器发送事件用于 Facebook/Twitter 更新、估价更新、新的博文、赛事结果等。

1. 浏览器支持

除了 Internet Explorer，所有主流浏览器均支持服务器发送事件。

2. 接收 Server-Sent 事件通知

EventSource 对象用于接收服务器发送事件通知，例如：

```
var source=new EventSource("例 8-2-6-1.aspx");
source.onmessage=function(event)
{
    document.getElementById("result").innerHTML+=event.data+"<br>";
};
```

创建一个新的 EventSource 对象，规定发送更新页面的 URL("demo_sse.php")。每接收到一次更新，就会发生 onmessage 事件。当 onmessage 事件发生时，把已接收的数据推入 id 为 result 的元素中。

3. 检测 Server-Sent 事件支持

以下程序代码用来检测浏览器是否支持服务器发送事件。

```
if(typeof(EventSource)!=="undefined")
{
    //浏览器支持 Server-Sent
    //一些代码.....
}
else
{
//浏览器不支持 Server-Sent..
}
```

4. 一个完整的实例

例 8-2-6-1 利用 HTML5 服务器事件，每 5 秒刷新一次网页时间。

(1) 浏览器端代码：例 8-2-6-1. htm

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
        <title></title>
    </head>
    <body>
        <h1>获取服务端更新数据</h1>
        <div id="result"></div>
        <script>
            if (typeof (EventSource) !=="undefined") {
                var source=new EventSource("例 8-2-6-1.aspx");
                source.onmessage=function (event) {
                    document.getElementById("result").innerHTML+=event.data+"<br>";
                };
            }
            else {
                document.getElementById("result").innerHTML="抱歉,你的浏览器不支持
                server-sent 事件...";
            }
        </script>
    </body>
</html>
```

(2) 服务器端代码

为了例 8-2-6-1. htm 浏览器端代码可以运行，需要服务器能够定时发送数据更新。服务器端事件流的语法要求把 Content-Type 报头设置为 text/event-stream。例 8-2-6-1 . aspx完整的 ASPX 文件内容如下。

① 例 8-2-6-1. aspx 源文档。

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="例 8-2-6-1.aspx.cs" Inherits=
"理论_第 8 章_HTML5_例 8_2_6_1" %>
<%Response.ContentType="text/event-stream";
    Response.Expires=-1;
    Response.Write("data: "+DateTime.Now);
    Response.Flush();
%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head runat="server"><title></title></head>
    <body><form id="form1" runat="server"><div></div></form></body>
</html>
```

首先把报头 Content-Type 设置为 text/event-stream，规定不对页面进行缓存，输出发送日期(始终以 data：开头，data 是键名，其值是时间，这与 event. data 一致)，最后向网页刷新输出数据。

② 例 8-2-6-1. aspx. cs 源文档。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
public partial class 理论_第 8 章_HTML5_例 8_2_6_1 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e) {   }
}
```

(3) 浏览器显示结果

浏览器显示结果如图 8. 15 所示。

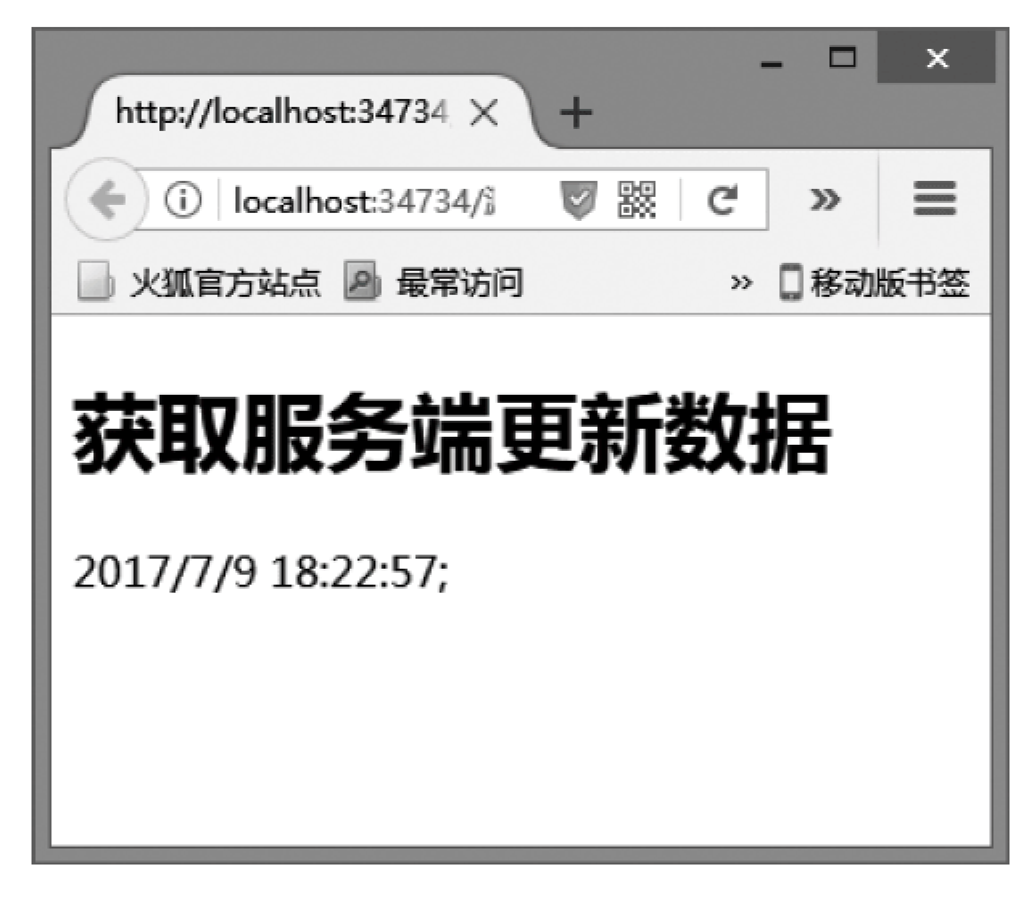


图 8. 15　例 8-2-6-2 的浏览器显示结果

5. EventSource 对象

例 8-2-6-2　使用 onmessage 事件获取消息。EventSource 对象还有其他事件，如下所示。

① onopen：当通往服务器的连接被打开。

② onerror：当发生错误。

8.2.7 WebSocket

WebSocket 是 HTML5 开始提供的一种在单个 TCP 连接上进行全双工通信的协议。在 WebSocket API 中,浏览器和服务器只需一个握手动作,然后浏览器和服务器之间就形成了一条快速通道。两者之间就可以直接进行数据互相传送。

浏览器通过 JavaScript 向服务器发出建立 WebSocket 连接的请求,连接建立后,客户端和服务器端就可以通过 TCP 连接直接交换数据。当程序获取到 Web Socket 连接后,就可以通过 send()方法向服务器发送数据,并通过 onmessage 事件接收服务器返回的数据。

用于创建 WebSocket 对象的 API 如下。

```
var Socket=new WebSocket(url, [protocol]);
```

参数说明如下。

① url:指定连接的 URL。

② protocol:可选的,指定可接受的子协议。

1. WebSocket 属性

WebSocket 对象属性如下。

① readyState:只读属性,表示连接状态,枚举值 0 表示连接尚未建立,1 表示连接已建立,可以进行通信,2 表示连接正在进行关闭,3 表示连接已经关闭或者连接不能打开。

② bufferedAmount:只读属性,已被 send() 放入正在队列中等待传输但还没有发出的 UTF-8 文本字节数。

2. WebSocket 事件

WebSocket 相关事件如表 8.2 所示。

表 8.2 WebSocket 相关事件

事 件	关联方法	描 述
open	onopen	连接建立时触发
message	onmessage	客户端接收服务端数据时触发
error	onerror	通信发生错误时触发
close	onclose	连接关闭时触发

3. WebSocket 方法

WebSocket 相关方法如下。

① send():使用连接发送数据。

② close():关闭连接。

4. WebSocket 实例

WebSocket 协议本质上是一个基于 TCP 的协议。为建立一个 WebSocket 连接，客户端浏览器首先要向服务器发起一个 HTTP 请求，这个请求和通常的 HTTP 请求不同，包含了一些附加头信息，其中附加头信息 Upgrade：WebSocket 表明这是一个申请协议升级的 HTTP 请求。服务器端解析这些附加的头信息，然后产生应答信息返回给客户端，客户端和服务器端的 WebSocket 连接就建立起来了，双方就可以通过这个连接通道自由传递信息，并且这个连接会持续存在，直到客户端或者服务器端的某一方主动地关闭连接。

例 8-2-7-1 一个 WebSocket 应用实例。

（1）浏览器

目前，大部分浏览器支持 WebSocket（）接口，可以在 Chrome、Firefox、Opera 和 Safari 浏览器中浏览例 8-2-7-1.htm，源文档如下。

```
<!DOCTYPE HTML>
<html>
  <head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>菜鸟教程(runoob.com)</title>
      <script type="text/javascript">
         function WebSocketTest()
         {
            if ("WebSocket" in window)
            {
               alert("您的浏览器支持 WebSocket!");
               //打开一个 web socket
               var ws=new WebSocket("ws://localhost:9998/echo");
               ws.onopen=function()
               {
                  //Web Socket 已连接上,使用 send()方法发送数据
                  ws.send("发送数据");
                  alert("数据发送中...");
               };
               ws.onmessage=function (evt)
               {
                  var received_msg=evt.data;
                  alert("数据已接收...");
               };
               ws.onclose=function()
               {
                  //关闭 websocket
                  alert("连接已关闭...");
```

```
                };
            }
            else
            {
                //浏览器不支持 WebSocket
                alert("您的浏览器不支持 WebSocket!");
            }
        }
    </script>
  </head>
  <body>
      <div id="sse">
        <a href="javascript:WebSocketTest()">运行 WebSocket</a>
      </div>
  </body>
</html>
```

(2) 配置 Apache 服务器

① 安装 pywebsocket。

浏览例 8-2-7-1.htm 前,需要创建一个支持 WebSocket 的服务。访问如下网址:

```
https://github.com/google/pywebsocket
```

而后单击 Clone or Download 按钮进行下载。

② 解压下载的文件。

③ 进入 pywebsocket 目录。

④ 执行如下命令。

```
$python setup.py build
$sudo python setup.py install
$pydoc mod_pywebsocket
```

⑤ 开启服务。

在 pywebsocket/mod_pywebsocket 目录下执行以下命令。

```
$sudo python standalone.py -p 9998 -w ../example/
```

以上命令会开启一个端口号为 9998 的服务,使用 -w 来设置处理程序 echo_wsh.py 所在的目录。

8.3 上机实验样例

1. 实验目标

综合运用定时器、画布和 Web 存储进行网页动画程序开发。

2. 程序功能

运用画布，在浏览器内输出一个圆，周而复始地从左向右、从右向左水平匀速运动。具体要求如下。

① 圆的定时运动用一个对象刻画。

② 默认速度为频率为每秒运动 10 次，每次步长为 2px。

③ 用户可设定运动速度，分频率和步长 2 个方面。

④ 设置的参数用 Web 存储技术记录用户参数。

⑤ 按 R 键，圆的填充色变红，按 G 键，圆的填充色变绿。

⑥ 当圆碰到画布的边框时，改变运动方向。

3. 实验步骤

（1）源程序：8-3. htm

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
        <script src="jquery-1.3.2.min.js" type="text/javascript">
        </script>
        <script type="text/javascript">
            var Ball={
                X: 10, Y: 10, R: 10, D: 1, Cxt: null,
                Initiation: function () {
                    Ball.Cxt=document.getElementById("can1").getContext("2d");
                    Ball.Cxt.fillStyle="#FF0000";
                    if (typeof (Storage) !=="undefined") {
                        if (localStorage.Frequency) return;
                        localStorage.Frequency=10;
                        localStorage.Step=10;
                    }
                    else {
                        $("#Prompt").html("对不起,您的浏览器不支持 Web 存储。");
                    }
                },
                Change: {
                    Color: function (event) {
                        switch (event.keyCode) {
                            case 82: Ball.Cxt.fillStyle="#FF0000"; break;
                            case 71: Ball.Cxt.fillStyle="#00FF00"; break;
                        }
                    },
```

```
                Frequency: function () {
                    localStorage.Frequency=$("#Frequency").val();
                    if (Ball.Play.Timer !=null) { Ball.Play.End(); Ball.Play.Start(); }
                },
                Step: function () { localStorage.Step=$("#Step").val(); }
            },
            Render: function () {
                Ball.Cxt.beginPath();
                Ball.Cxt.arc(Ball.X, Ball.Y, Ball.R, 0, Math.PI * 2, true);
                Ball.Cxt.closePath();
                Ball.Cxt.fill();
            },
            Move: function () {
                Ball.Cxt.clearRect(0, 0, 250, 250);
                Ball.X=Ball.X+Number(localStorage.Step) * Ball.D;
                Ball.Render();
                if (Ball.X<=Ball.R || Ball.X==250-Ball.R) Ball.D=Ball.D * (-1);
            },
            Play: {
                Timer: null,
                Start: function () {
                    if (Ball.Play.Timer==null)
                        Ball.Play.Timer=setInterval(Ball.Move, 1000/Number
                        (localStorage.Frequency));
                },
                End: function () { clearInterval(Ball.Play.Timer); Ball.
                Play.Timer=null; }
            }
        };
        $(document).ready(
            function () {
                $("body").keydown(function (event) { Ball.Change.Color
                (event); });
                Ball.Initiation();
                Ball.Render();
                Ball.Play.Start();
                $("#Frequency").val(localStorage.Frequency);
                $("#Step").val(localStorage.Step);
            });
    </script>
</head>
<body>
    <p>按 R 键填充色为红,按 G 键填充色为绿</p>
    <div>
```

```
            <input  type='button' id="continue" value="继续" onclick='Ball.
            Play.Start();'/>
            <input type='button' id="Stop" value="停止" onclick='Ball.Play.
            End();'/>
            <label>速度:</label><select id="Frequency" onchange="Ball.
            Change.Frequency();">
                                    <option value="10">10</option>
                                    <option value="20">20</option>
                                    <option value="40">40</option>
                                </select>
            <label>次/秒,</label>
            <label>步长:</label>
                        <select id="Step" onchange="Ball.Change.Step();">
                            <option value="10">10</option>
                            <option value="5">5</option>
                            <option value="2">2</option>
                        </select>
            <label>px</label>
        </div>
        <canvas id="can1" width="250" height="250" style="background-color:#bd92a3">
        </canvas>
        <div id="Prompt"></div>
    </body>
</html>
```

(2) Ball 对象设计

Ball 为一个对象,用于控制一个画布内圆的水平匀速运动。该对象有 5 个属性、2 个方法、1 个子对象。

① 5 个属性。

- Ball. X:矩形左上角横坐标。
- Ball. Y:矩形左上角纵坐标。
- Ball. D:运动方向,取值 1 表示从左向右,取值－1 表示从右向左。
- Ball. R:圆的半径。
- Ball. Cxt:绘图环境变量。

② 3 个方法。

- Ball. Initiaition():初始化函数,设置画布绘图环境变量、运动频率和运动步长。
- Ball. Move():根据 Ball. X、Ball. Y、当前运动方向以及步长,让圆动一步,更新当前 Ball. X 和 Ball. Y,清屏,调用 Ball. Renbder()画圆 1 次。如果当前圆对象水平坐标为 250- Ball. R,将运动方向设为从右向左,即 1。如果当前矩形水平坐标为 Ball. R,将运动方向设为从左向右,即－1。
- Ball. Renbder():根据当前 Ball. X 和 Ball. Y 画圆 1 次。

③ 2 个子对象。

• Ball. Play 子对象。

Ball. Play 子对象具有 1 个属性和 2 个方法，即 Ball. Play. Timer 属性、Ball. Play. Start()和 Ball. Play. End()。

Ball. Play. Timer：存储定时器，初始值为空。

Ball. Play. Start()：如果 Ball. Play. Timer 为空，则用 setInterval()方法创建定时器。setInterval()的定时间隔为(1000 / Number(localStorage. Frequency)毫秒，所执行的方法为 Ball. Move()。

• Ball. Chang 子对象。

Ball. Chang 子对象用于集成 3 个方法：Ball. Chang. Color()、Ball. Chang. Frequency()和 Ball. Chang. Step()。这 3 个方法用于改变颜色、改变运动频率和改变运动步长，分别对应键盘按下事件、id 为 Frequency 的 select 选择改变事件、id 为 Step 的 select 选择改变事件的处理函数。

(3) 文档就绪事件处理

在文档就绪事件处理函数中进行 body 键盘按键事件绑定、Ball 对象初始化、Ball 对象直接画圆 1 次、启动定时器、根据 Web 存储初始化运动频率和运动速度的操作。

(4) 浏览器显示结果

浏览器显示结果如图 8.16 所示。



图 8.16　8.3 上机实验的浏览器显示结果

(5) 思考题

① 在图 8.16 的状态下，将运动速度改为 20 次/秒，步长设为 10px，小球运动明显加快。单击浏览器的刷新按钮，根据什么现象说明 Web 存储已经起作用了？

② 在上一步的基础上关闭浏览器，重新用同一浏览器浏览 8-3. htm，凭什么说明

Web 存储已经起作用了?

③ 在上一步的基础上重新启动计算机,重新用同一浏览器浏览 8-3.htm,凭什么说明 Web 存储已经起作用了?

④ 将运动小球圆心的纵向坐标改为画布的纵向中心点,如何更改?

⑤ 如何将界面做得更精致美观?

8.4 实验任务

1. 实验题目

运用画布、定时器、jQuery 和 Web 存储开发面向对象动画脚本程序。

2. 程序功能

在实验 6.5 的基础上新增如下功能。

(1) 在纵向树形面板上新增速度选项,内含运动频率和运动步长 2 个方面,并编程增加圆的静和动 2 个功能。当在树形面板中选中动,主菜单中选中圆,则圆从左上角运动到右下角,碰到右下角而反向运动到左上角,碰到左上角后再次反向运动,周而复始,直到静按钮被按下。

(2) 左侧树形面板选择全部用 Web 存储记录,打开网页后用 Web 存储初始化当前选项。

3. 实验目的

(1) 掌握运用 JSON 进行面向对象程序设计的方法。
(2) 深化动画实现机制的理解。
(3) 巩固 jQuery 事件的运用方法。
(4) 巩固 jQuery CSS 改变元素样式的方法。
(5) 掌握画布绘图技术。
(6) 掌握 Web 存储技术的应用。

4. 实验类型

综合设计。

5. 实验要求

(1) 脚本程序完全运用 jQuery 程序实现。
(2) 运用 JSON 进行面向对象的应用程序开发。
(3) 所有画图全部用画布技术实现。
(4) 左侧树形面板的选择用 Web 存储初始化和记录。

6. 实验环境

(1) 计算机：PC、内存 8GB、主频 1.8GHz 及以上、硬盘 500GB 及以上。

(2) 操作系统：Windows XP、Windows 7、Windows 8、Windows 10。

(3) 开发环境：Visual Studio 2010。

(4) 浏览器：IE9 及以上、Chrome、Firefox、Safari、Edge、QQ 浏览器等。

7. 实验原理

给出与事件、CSS 添加删除元素直接相关的 jQuery 语句，并辅以必要的说明与分析。

8. 遇到的问题及解决办法

(1) 给出所遇到的全部错误现象描述。

(2) 给出修正错误前设定的断点位置。

(3) 给出修正错误所监视的变量。

(4) 给出运行到断点处监视变量值。

9. 运行结果

给出所完成任务功能的屏幕截图。

第 9 章



jQuery Mobile

■ **知识目标**

- 掌握手机端布局技术
- 掌握界面元素——工具栏、导航栏、按钮、图标、列表的应用
- 掌握手机端事件关联及其处理技术
- 掌握 IIS 服务器发布 Web 网站相关技术

■ **能力目标**

- 能够根据实际需求进行恰当布局
- 能够根据实际需求选取合适的界面静态元素和界面交互机制
- 能够部署 IIS Web 服务器

■ **素质目标**

- 快速开发体验度良好的手机端网页

■ **教学重点**

- jQuery Mobile 布局技术和界面交互元素

■ **教学难点**

- 在手机屏幕上恰当地布局

■ **建议学时**

- 理论：4 学时
- 实验：4 学时

9.1 基 础 知 识

jQuery Mobile 是一个为触控优化的框架，用于创建移动 Web 应用程序，适用于所有流行的智能手机和平板电脑，使用 HTML5 和 CSS3 可以通过尽可能少的脚本对页面进行布局。jQuery Mobile 构建于 jQuery 库之上，这使得已掌握 jQuery 的读者更易学习。它使用 HTML5、CSS3、JavaScript 和 Ajax，通过尽可能少的代码来完成页面的布局。

jQuery Mobile 将“写得更少、做得更多”这一理念提升到了新的层次：它会自动为网

页设计交互的易用外观，并在所有移动设计上保持一致。这样不需要为每种移动设备或OS编写一个应用程序。

① Android 和 Blackberry 用 Java 编写。

② iOS 用 Objective C 编写。

③ Windows Phone 用 C#和.net 编写。

jQuery Mobile 解决了这个问题，因为它只用 HTML、CSS 和 JavaScript，这些技术是所有移动 Web 浏览器的标准。尽管 jQuery Mobile 工作于所有移动设备，但也可能在桌面计算机上存在兼容性问题。

9.1.1 jQuery Mobile 安装

在网站上使用 jQuery Mobile 有以下 2 种方法。

① 从 CDN 引用 jQuery Mobile(推荐)。

② 从 jQuerymobile.com 下载 jQuery Mobile 库。

(1) 从 CDN 引用 jQuery Mobile

CDN (Content Delivery Network)用于通过 Web 来分发常用的文件，以此加快下载速度。与 jQuery 类似，无须在计算机上安装任何程序，只需直接在 HTML 页面中引用以下样式表和 JavaScript 库。

```
<head>
<link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.
min.css">
<script src="http://code.jquery.com/jquery-1.8.3.min.js"></script>
<script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.js"></
script>
</head>
```

(2) 下载 jQuery Mobile 到 Web 应用服务器

可以从 jQuerymobile.com 下载文件，而后在服务器上存放 jQuery Mobile，如下所示。

```
<head>
<link rel=stylesheet href=jquery.mobile-1.3.2.css>
<script src=jquery.js></script>
<script src=jquery.mobile-1.3.2.js></script>
</head>
```

说明：将下载的文件放到目标文件夹中，需要先创建文件夹。<script> 标签中没有 type="text/javascript" 属性，因为在 HTML5 中，该属性不是必需的。JavaScript 是 HTML5 以及所有现代浏览器中的默认脚本语言。

9.1.2 jQuery Mobile 页面

尽管 jQuery Mobile 适用于所有移动设备，它在台式计算机上仍然可能存在兼容性

问题(由于有限的 CSS3 支持)。

例 9-1-2-1 一个简单的 jQuery Mobile 页面。

(1) 例 9-1-2-1. htm 源文档

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.mobile
        -1.3.2.min.css">
        <script src="http://code.jquery.com/jquery-1.8.3.min.js"></script>
        <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.js">
        </script>
    </head>
    <body>
        <div data-role="page">
            <div data-role="header">
              <h1>欢迎访问我的主页</h1>
            </div>
            <div data-role="content">
              <p>现在我已经成为一名移动开发者!</p>
            </div>
            <div data-role="footer"><h1>页脚文本</h1></div>
        </div>
    </body>
</html>
```

(2) Android 平台小米手机浏览器显示结果

Android 平台小米手机浏览器显示结果如图 9.1 所示。



图 9.1 例 9-1-2-1 的 Android 平台小米手机浏览器显示结果

(3) 程序解析

① data-role="page" 是显示在浏览器中的页面。

② data-role="header" 创建页面上方的工具栏(常用于标题和搜索按钮)。

③ data-role="content" 定义页面的内容,比如文本、图像、表单和按钮等。

④ data-role="footer" 创建页面底部的工具栏。

HTML5 的 data-* 属性用于通过 jQuery Mobile 为移动设备创建“对触控友好的”交互外观。在以上 4 类容器中,可以添加任意 HTML 元素,如段落、图像、标题、列表等。

1. 在 jQuery Mobile 中添加页面

用 jQuery Mobile 可以在单一 HTML 文件中创建多个页面,通过唯一的 id 分隔每张页面,并使用 href 属性连接彼此,如例 9-1-2-2 所示。

例 9-1-2-2 jQuery Mobile 多网页的简单实例。

(1) 例 9-1-2-2.htm 源文档

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.mobile
        -1.3.2.min.css">
        <script src="http://code.jquery.com/jquery-1.8.3.min.js"></script>
        <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.js">
        </script>
    </head>
    <body>
        <div data-role="page" id="pageone">
          <div data-role="header"><h1>欢迎访问我的主页</h1></div>
          <div data-role="content"><p>Welcome!</p><a href="#pagetwo">转到页面二</
          a></div>
          <div data-role="footer"><h1>页脚文本</h1></div>
        </div>

        <div data-role="page" id="pagetwo">
          <div data-role="header"><h1>欢迎访问我的主页</h1></div>
          <div data-role="content"><p>Goodbye!</p><a href="#pageone">转到页面一</
          a></div>
          <div data-role="footer"><h1>页脚文本</h1>
          </div>
        </div>
    </body>
</html>
```

(2) Android 平台小米手机浏览器显示结果

Android 平台小米手机浏览器显示结果如图 9.2 所示。



图 9.2　例 9-1-2-2 的 Android 平台小米手机浏览器显示结果

(3) 提示

包含大量内容的 Web 应用程序会影响加载时间。如不希望在内部链接页面，可使用如下语句引用外部文件。

```
<a href="externalfile.html">转到外部页面</a>
```

2. 将页面用做对话框

对话框是用来显示信息或请求输入的视窗类型。需在用户单击(轻触)链接时创建一个对话框，向该链接添加 data-rel="dialog"，如例 9-1-2-3 所示。

例 9-1-2-3　一个简单的页面对话框实例。

(1) 例 9-1-2-3.htm 源文档

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.mobile
        -1.3.2.min.css">
        <script src="http://code.jquery.com/jquery-1.8.3.min.js"></script>
        <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.js">
       </script>
    </head>
    <body>
        <div data-role="page" id="pageone">
              <div data-role="header"><h1>欢迎访问我的主页</h1></div>
              <div data-role="content">
                <p>欢迎！</p>
                <a href="#pagetwo" data-rel="dialog">转到页面二</a>
              </div>
              <div data-role="footer"><h1>页脚文本</h1></div>
        </div>
```

```
    <div data-role="page" id="pagetwo">
      <div data-role="header"><h1>我是一个对话框！</h1></div>
      <div data-role="content">
            <p>对话框显示在当前页面的顶端,不横跨整个页面宽度。对话框页眉中的图
            标"X"可关闭对话框。</p>
            <a href="#pageone">转到页面一</a>
      </div>
      <div data-role="footer"><h1>页脚文本</h1></div>
    </div>
</body>
```

(2) Android 平台小米手机浏览器显示结果

Android 平台小米手机浏览器显示结果如图 9.3 和图 9.4 所示。图 9.3 和图 9.4 分别为未单击“转到页面二”链接和单击“转到页面二”链接后的界面。单击图 9.4 对话框的页眉左侧图标按钮,关闭对话框,退到图 9.3 的状态。



图 9.3　例 9-1-2-3 的 Android 平台小米手机浏览器显示结果(未单击“转到页面二”链接)



图 9.4　例 9-1-2-3 的 Android 平台小米手机浏览器显示结果(单击“转到页面二”链接后)

9.1.3 jQuery Mobile 按钮

移动应用程序应该触控便利，这需要按钮的支持。jQuery Mobile 能创建按钮，有以下 3 种方法创建。

① 使用<button>元素。

② 使用<input>元素。

③ 使用 data-role="button" 的<a>元素。

1. <button>

<button>元素的使用如例 9-1-3-1 所示。

例 9-1-3-1 <button>元素使用的简单示例。

(1) 例 9-1-3-1.htm 源文档

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.mobile
        -1.3.2.min.css">
        <script src="http://code.jquery.com/jquery-1.8.3.min.js"></script>
        <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.js">
        </script>
    </head>
    <body>
        <div data-role="page" id="pageone">
          <div data-role="header"><h1>体验按钮</h1></div>
          <div data-role="content"><button>按钮</button></div>
          <div data-role="footer"><h1>页脚文本</h1></div>
        </div>
    </body>
</html>
```

(2) Android 平台小米手机浏览器显示结果

Android 平台小米手机浏览器显示结果如图 9.5 所示。

2. <input>

使用<input>元素的语法为：<input type="button" value="按钮">，一个简单的运用如例 9-1-3-2 所示。

例 9-1-3-2 使用<input>元素作为按钮的简单示例。

(1) 例 9-1-3-2.htm 源文档

```
<!DOCTYPE html>
```



图 9.5　例 9-1-3-1 的 Android 平台小米手机浏览器显示结果

```
<html>
    <head>
        <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.mobile
        -1.3.2.min.css">
        <script src="http://code.jquery.com/jquery-1.8.3.min.js"></script>
        <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.js">
       </script>
    </head>
    <body>
        <div data-role="page" id="pageone">
          <div data-role="header"><h1>input 元素作为按钮的示例</h1></div>
          <div data-role="content"><input type="button" value="按钮"></div>
          <div data-role="footer"><h1>页脚文本</h1></div>
        </div>
    </body>
</html>
```

(2) Android 平台小米手机浏览器显示结果

Android 平台小米手机浏览器显示结果如图 9.6 所示。

3. <a>

jQuery Mobile 中的按钮会自动获得样式,这增强了它们在移动设备上的交互性和可用性。建议使用 data-role="button"的<a>元素来作为导航按钮,创建页面之间的链接,而<input>或<button>元素用于表单提交。

(1) 块级按钮

超链接的块级按钮语法如下。



图 9.6　例 9-1-3-2 的 Android 平台小米手机浏览器显示结果

```
<a href="#" data-role="button">按钮</a>
```

超链接的使用如例 9-1-3-3 所示。

例 9-1-3-3　超链接按钮的简单示例。

① 例 9-1-3-3.htm 源文档。

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.mobile
        -1.3.2.min.css">
        <script src="http://code.jquery.com/jquery-1.8.3.min.js"></script>
        <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.js">
        </script>
    </head>
    <body>
        <div data-role="page" id="pageone">
            <div data-role="header"><h1>超链接作为按钮的示例</h1></div>
            <div data-role="content"><a href="#" data-role="button">按钮</a></
            div>
            <div data-role="footer"><h1>页脚文本</h1></div>
        </div>
    </body>
</html>
```

② Android 平台小米手机浏览器显示结果。

Android 平台小米手机浏览器显示结果如图 9.7 所示。



图 9.7 例 9-1-3-3 的 Android 平台小米手机浏览器显示结果

(2) 行内按钮

默认情况下,按钮会占据屏幕的全部宽度。如果需要按钮适应其内容,或者需要两个或多个按钮并排显示,可添加 data-inline="true",如例 9-1-3-4 所示。

例 9-1-3-4 行内按钮的简单示例。

① 例 9-1-3-4.htm 源文档。

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.mobile
        -1.3.2.min.css">
        <script src="http://code.jquery.com/jquery-1.8.3.min.js"></script>
        <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.js">
       </script>
    </head>
    <body>
        <div data-role="page" id="pageone">
          <div data-role="header"><h1>超链接导航行内按钮的示例</h1></div>
          <div data-role="content"><p>欢迎使用超链接导航</p>
                <a href="#pagetwo" data-role="button" data-inline="true">转到页面二
                </a>
          </div>
          <div data-role="footer"><h1>页脚文本</h1></div>
        </div>
        <div data-role="page" id="pagetwo">
          <div data-role="header"><h1>欢迎来到我的主页</h1></div>
          <div data-role="content">
              <p>再见!</p>
```

```
                <a href="#pageone" data-role="button" data-inline="true">转到页面一</
                a>
                <a href="#pageone" data-role="button" data-inline="true">转到页面一</
                a>
                <a href="#pageone" data-role="button" data-inline="true">转到页面一</
                a>
            </div>
            <div data-role="footer"><h1>页脚文本</h1></div>
        </div>
    </body>
</html>
```

② Android 平台小米手机浏览器显示结果。

单击“转到页面二”按钮前后的界面如图 9.8 和图 9.9 所示。



图 9.8　例 9-1-3-4 的 Android 平台小米手机浏览器显示结果(未单击“转到页面二”按钮)



图 9.9　单击图 9.8 的“转到页面二”按钮后 Android 平台小米手机浏览器显示结果

(3) 组合按钮

jQuery Mobile 提供了对按钮进行组合的简单方法。将 data-role="controlgroup" 属性与 data-type="horizontal|vertical" 一同使用,以规定水平或垂直地组合按钮,如例 9-1-3-5 所示。

例 9-1-3-5 组合按钮的简单示例。

① 例 9-1-3-5.htm 源文档。

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.mobile
        -1.3.2.min.css">
        <script src="http://code.jquery.com/jquery-1.8.3.min.js"></script>
        <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.js">
        </script>
    </head>
    <body>
        <div data-role="page" id="pageone">
            <div data-role="header"><h1>分组按钮</h1></div>
            <div data-role="content">
                <div data-role="controlgroup" data-type="horizontal">
                    <p>水平分组:</p>
                    <a href="#" data-role="button">按钮 1</a>
                    <a href="#" data-role="button">按钮 2</a>
                    <a href="#" data-role="button">按钮 3</a>
                </div><br>
                <div data-role="controlgroup" data-type="vertical">
                    <p>垂直分组(默认):</p>
                    <a href="#" data-role="button">按钮 1</a>
                    <a href="#" data-role="button">按钮 2</a>
                    <a href="#" data-role="button">按钮 3</a>
                </div>
            </div>
            <div data-role="footer"><h1>页脚文本</h1></div>
        </div>
    </body>
</html>
```

② Android 平台小米手机浏览器显示结果。

Android 平台小米手机浏览器显示结果如图 9.10 所示。

③ 说明。

默认情况下,组合按钮是垂直分组的,彼此间没有外边距和空白。只有第一个和最后一个按钮拥有圆角,组合后就创造出了漂亮的外观。

图 9.10　例 9-1-3-5 的 Android 平台小米手机浏览器显示结果

(4) 后退按钮

创建后退按钮，需使用 data-rel="back" 属性（会忽略锚的 href 值），如例 9-1-3-6 所示。

例 9-1-3-6　后退按钮的简单示例。源文档如下。

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.mobile
        -1.3.2.min.css">
        <script src="http://code.jquery.com/jquery-1.8.3.min.js"></script>
        <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.js">
        </script>
    </head>
    <body>
        <div data-role="page" id="pageone">
            <div data-role="header"><h1>后退按钮实例</h1></div>
            <div data-role="content">
                <a href="#pagetwo" data-role="button">转到页面二</a>
            </div>
            <div data-role="footer"><h1>页脚文本</h1></div>
        </div>
        <div data-role="page" id="pagetwo">
            <div data-role="header"><h1>后退按钮实例</h1></div>
            <div data-role="content">
                <a href="#" data-role="button" data-rel="back">后退</a>
            </div>
            <div data-role="footer"><h1>页脚文本</h1></div>
```

```
        </div>
    </body>
</html>
```

9.1.4 按钮图标

jQuery Mobile 提供的一套图标可使按钮更具吸引力。向按钮添加图标，使用 data-icon 属性，如下所示。

```
<a href="#anylink" data-role="button" data-icon="search">搜索</a>
```

除了超链接外，也可以在<button>或<input>元素中使用 data-icon 属性。jQuery Mobile 提供的一些可用图标如表 9.1 所示。

表 9.1 data-icon 属性值及其描述

属 性 值	描 述	属 性 值	描 述
data-icon="arrow-l"	左箭头	data-icon="home"	首页
data-icon="arrow-r"	右箭头	data-icon="back"	返回
data-icon="delete"	删除	data-icon="search"	搜索
data-icon="info"	信息	data-icon="grid"	网格

图标被放置的位置有上、右、下或左，使用 data-iconpos 属性来规定位置，如下所示。

```
<a href="#link" data-role="button" data-icon="search" data-iconpos="top">上</a>
<a href="#link" data-role="button" data-icon="search" data-iconpos="right">右</a>
<a href="#link" data-role="button" data-icon="search" data-iconpos="bottom">下</a>
```

默认地，所有按钮中的图标靠左放置。如果只需显示图标，将 data-iconpos 设置为"notext"，如下所示。

```
<a href="#link" data-role="button" data-icon="search" data-iconpos="notext">搜索</
a>
```

9.1.5 工具栏

工具栏元素常被放置于页眉和页脚中，以实现"已访问"的导航。

1. 标题栏

页眉通常包含页眉标题/LOGO 或 3 个按钮(通常是首页、选项或搜索按钮)。可以在页眉中向左侧或/以及右侧添加按钮。在页眉标题文本的左侧添加 1 个按钮，在右侧添加 1 个按钮，如例 9-1-5-1 所示。

例 9-1-5-1 页眉具有左右 2 个按钮的标题栏示例。

(1) 相关代码

```
<div data-role="header">
    <a href="#" data-role="button">首页</a>
    <h1>欢迎来到我的主页</h1>
    <a href="#" data-role="button">搜索</a>
</div>
```

(2) Android 平台小米手机浏览器显示结果

Android 平台小米手机浏览器显示结果如图 9.11 所示。



图 9.11　例 9-1-5-1 的 Android 平台小米手机浏览器显示结果

(3) 代码解析

下面的代码向页眉标题的左侧添加 1 个按钮。

```
<div data-role="header">
    <a href="#" data-role="button">首页</a>
    <h1>欢迎来到我的主页</h1>
</div>
```

如在<h1>元素之后放置按钮链接，它不会显示在文本右侧。如需向页眉标题的右侧添加按钮，需要规定类名 ui-btn-right，如下所示。

```
<div data-role="header">
    <h1>欢迎来到我的主页</h1>
    <a href="#" data-role="button" class="ui-btn-right">搜索</a>
</div>
```

(4) 提示

页眉可包含 1 个或 2 个按钮,页脚没有限制。

2. 页脚栏

与页眉相比,页脚更具伸缩性,更实用且多变,能够包含所需数量的按钮,如例 9-1-5-2 所示。

例 9-1-5-2 页眉具有左右 2 个按钮的标题栏示例。

(1) 例 9-1-5-2. htm 源文档

```
<!DOCTYPE html>
<html>
    <head>
    <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.
    3.2.min.css">
    <script src="http://code.jquery.com/jquery-1.8.3.min.js"></script>
    <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.js"></
    script>
    </head>
    <body>
        <div data-role="page">
            <div data-role="header">
                <a href="#" data-role="button" data-icon="home">首页</a>
                <h1>欢迎访问我的主页</h1>
                <a href="#" data-role="button" data-icon="search">搜索</a>
            </div>
            <div data-role="content"><p>仅供演示标题栏和工具栏,无内容。</p></
            div>
            <div data-role="footer">
                <a href="#" data-role="button" data-icon="plus">转播到新浪微博
                </a>
                <a href="#" data-role="button" data-icon="plus">转播到腾讯微博
                </a>
                <a href="#" data-role="button" data-icon="plus">转播到 QQ 空间</
                a>
            </div>
        </div>
    </body>
</html>
```

(2) Android 平台小米手机浏览器显示结果

Android 平台小米手机浏览器显示结果如图 9.12 所示。



图 9.12　例 9-1-5-2 的 Android 平台小米手机浏览器显示结果

(3) 提示

页脚与页眉的样式不同，它会减去一些内边距和空白，并且按钮不会居中。如果要修正该问题，可在页脚设置类名 ui-btn。

```
<div data-role="footer" class="ui-btn">
```

能够在页脚中水平或垂直组合按钮进行选择，选择水平的示例如下所示。

```
<div data-role="footer" class="ui-btn">
    <div data-role="controlgroup" data-type="horizontal">
        <a href="#" data-role="button" data-icon="plus">转播到新浪微博</a>
        <a href="#" data-role="button" data-icon="plus">转播到腾讯微博</a>
        <a href="#" data-role="button" data-icon="plus">转播到 QQ 空间</a>
    </div>
</div>
```

3. 定位页眉和页脚

页眉和页脚使用 data-position 属性来定位，放置页眉和页脚的方式有以下 3 种。

① inline：默认页眉和页脚与页面内容位于行内。

② fixed：页面和页脚会留在页面顶部和底部。

③ fullscreen：与 fixed 类似，页面和页脚会留在页面顶部和底部，但浮于页面内容之上。

fullscreen 对于照片、图像和视频非常理想。对于 fixed 和 fullscreen 定位，触摸屏幕将隐藏和显示页眉及页脚。

(1) inline 定位(默认)

```
<div data-role="header" data-position="inline"></div>
```

```
<div data-role="footer" data-position="inline"></div>
```

（2）fixed 定位

```
<div data-role="header" data-position="fixed"></div>
<div data-role="footer" data-position="fixed"></div>
```

（3）fullscreen 定位

启用全面定位，使用 data-position＝"fixed"，并向该元素添加 data-fullscreen 属性。

```
<div data-role="header" data-position="fixed" data-fullscreen="true"></div>
<div data-role="footer" data-position="fixed" data-fullscreen="true"></div>
```

9.1.6 导航栏

1. 导航栏

导航栏由一组水平排列的链接构成，通常位于页眉或页脚内部。默认地，导航栏中的链接会自动转换为按钮。使用 data-role＝"navbar" 属性来定义导航栏，如例 9-1-6 所示。

例 9-1-6-1 导航栏使用的一个简单示例。

（1）例 9-1-6-1. htm 源文档

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.mobile
        -1.3.2.min.css">
        <script src="http://code.jquery.com/jquery-1.8.3.min.js"></script>
        <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.js">
        </script>
    </head>
    <body>
        <div data-role="page" id="pageone">
          <div data-role="header">
                <h1>欢迎来到我的主页</h1>
                <div data-role="navbar">
                      <ul>
                            <li><a href="#" data-icon="home">首页</a></li>
                            <li><a href="#" data-icon="arrow-r">页面二</a></li>
                            <li><a href="#" data-icon="search">搜索</a></li>
                      </ul>
                </div>
          </div>
          <div data-role="content"><p>我的内容..</p></div>
```

```
            <div data-role="footer">
                <h1>我的页脚</h1>
            </div>
        </div>
    </body>
</html>
```

(2) Android 平台小米手机浏览器显示结果

Android 平台小米手机浏览器显示结果如图 9.13 所示。

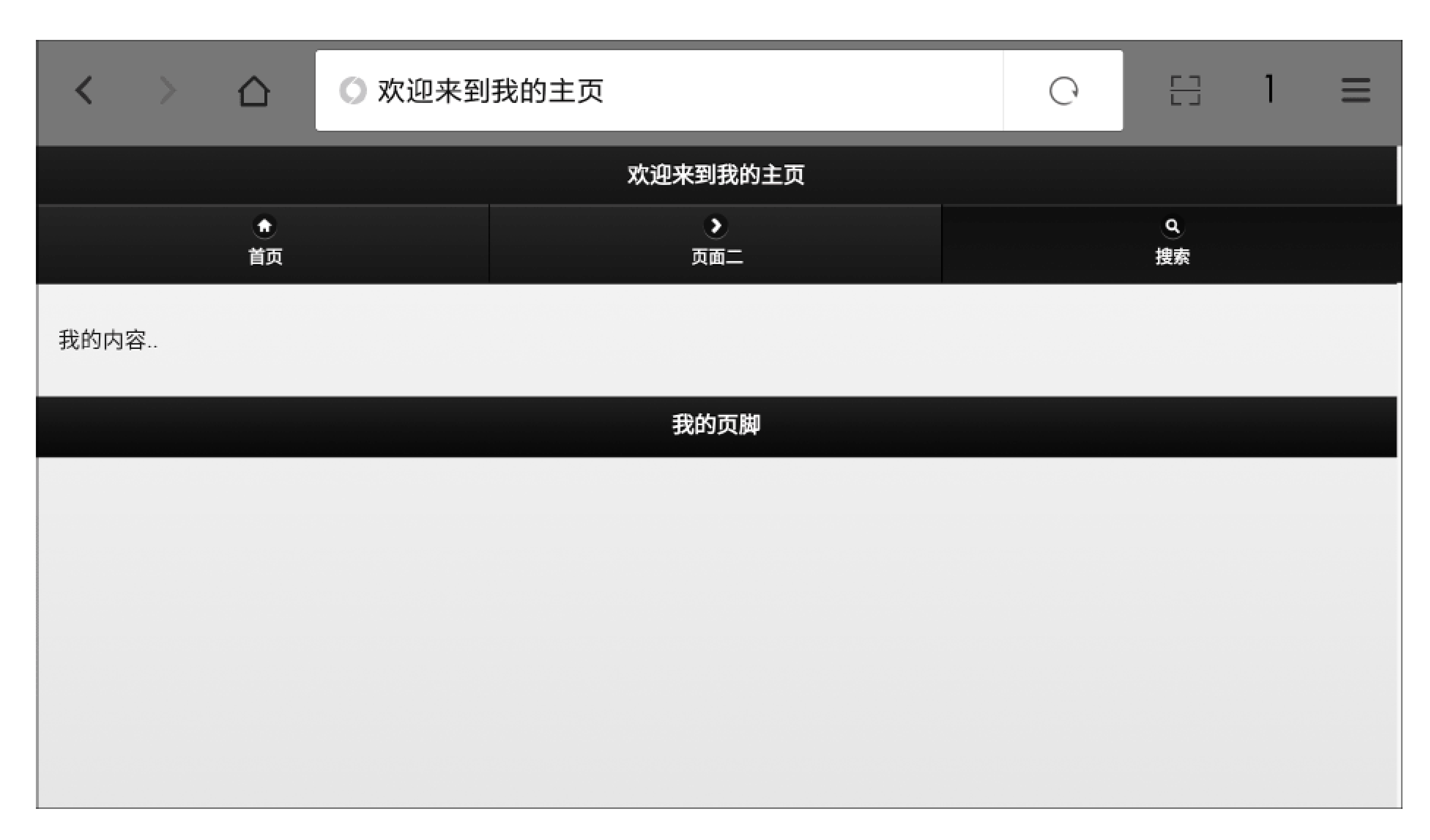


图 9.13　例 9-1-6-1 的 Android 平台小米手机浏览器显示结果

(3) 提示

按钮的宽度默认与其内容一致。使用无序列表来均等地划分按钮：1 个按钮占据 100% 的宽度，2 个按钮各 50%的宽度，3 个按钮各 33.3%的宽度，以此类推。如果在导航栏中规定了 5 个以上的按钮，它会弯折为多行。

2. 活动按钮

当单击导航栏中的链接时，会获得选取的外观。如需在不单击链接时实现此外观，请使用 class="ui-btn-active"：

```
<li><a href="#anylink" class="ui-btn-active">首页</a></li>
```

3. 内容中的导航栏

内容中的导航栏如例 9-1-6-2 所示。

例 9-1-6-2　内容中的导航栏示例。

(1) 例 9-1-6-2.htm 源文档

```
<!DOCTYPE html>
```

```
<html>
    <head>
        <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.mobile
        -1.3.2.min.css">
        <script src="http://code.jquery.com/jquery-1.8.3.min.js"></script>
        <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.js">
       </script>
    </head>
    <body>
        <div data-role="page" id="pageone">
            <div data-role="header"><h1>欢迎来到我的主页</h1></div>
            <div data-role="content">
                <div data-role="navbar">
                    <ul>
                        <li><a href="#" data-icon="plus">更多</a></li>
                        <li><a href="#" data-icon="minus">更少</a></li>
                        <li><a href="#" data-icon="delete">删除</a></li>
                        <li><a href="#" data-icon="check">喜爱</a></li>
                        <li><a href="#" data-icon="info">信息</a></li>
                    </ul>
                </div>
                <p>该例演示内容中的导航栏。</p>
            </div>
            <div data-role="footer"><h1>我的页脚</h1></div>
        </div>
    </body>
</html>
```

(2) Android 平台小米手机浏览器显示结果

Android 平台小米手机浏览器显示结果如图 9.14 所示。



图 9.14　例 9-1-6-2 的 Android 平台小米手机浏览器显示结果

4. 页脚中的导航栏

页脚中的导航栏如例 9-1-6-3 所示。

例 9-1-6-3 页脚中的导航栏示例。

(1) 例 9-1-6-3. htm 源文档

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.mobile
        -1.3.2.min.css">
        <script src="http://code.jquery.com/jquery-1.8.3.min.js"></script>
        <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.js">
        </script>
    </head>
    <body>
        <div data-role="page" id="pageone">
            <div data-role="header"><h1>欢迎来到我的主页</h1></div>
            <div data-role="content">
                <p>该例演示页脚中的导航栏。</p>
            </div>
            <div data-role="footer">
                <div data-role="navbar">
                    <ul>
                        <li><a href="#" data-icon="plus">更多</a></li>
                        <li><a href="#" data-icon="minus">更少</a></li>
                        <li><a href="#" data-icon="delete">删除</a></li>
                        <li><a href="#" data-icon="check">喜爱</a></li>
                        <li><a href="#" data-icon="info">信息</a></li>
                    </ul>
                </div>
            </div>
        </div>
    </body>
</html>
```

(2) Android 平台小米手机浏览器显示结果

Android 平台小米手机浏览器显示结果如图 9.15 所示。

9.1.7 可折叠

可折叠(Collapsibles)可隐藏或显示内容,对于存储部分信息很有用。



图 9.15 例 9-1-6-3 的 Android 平台小米手机浏览器显示结果

1. 创建可折叠块

如需创建可折叠的内容块，向目标容器分配 data-role="collapsible" 属性。在容器中添加 1 个标题元素，其后是需要扩展的任意 HTML 标记。

例 9-1-7-1 一个可折叠的网页示例。

(1) 例 9-1-7-1.htm 源文档

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.mobile
        -1.3.2.min.css">
        <script src="http://code.jquery.com/jquery-1.8.3.min.js"></script>
        <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.js">
        </script>
    </head>
    <body>
        <div data-role="page" id="pageone">
            <div data-role="header"><h1>可折叠块</h1></div>
            <div data-role="content">
                <div data-role="collapsible">
                    <h1>单击我-我可以折叠！</h1>
                    <p>我是可折叠的内容。</p>
                </div>
            </div>
            <div data-role="footer"><h1>页脚文本</h1></div>
        </div>
```

```
    </body>
</html>
```

(2) Android 平台小米手机浏览器显示结果

Android 平台小米手机浏览器显示结果如图 9.16 和图 9.17 所示。



图 9.16　例 9-1-7-1 的 Android 平台小米手机浏览器显示结果(折叠态)



图 9.17　例 9-1-7-1 的 Android 平台小米手机浏览器显示结果(展开态)

单击图 9.16 的加号图标,关闭的内容被展开,如图 9.17 所示。单击图 9.17 的减号图标,已展开的内容被关闭,如图 9.16 所示。

(3) 提示

默认地,该内容是关闭的。如需在页面加载时扩展内容,使用 data-collapsed="false"。

2. 可折叠块嵌套

可以嵌套可折叠内容块,折叠内容块可以被嵌套任意次数,示例如下所示。

```
<div data-role="collapsible">
    <h1>单击我-我可以折叠! </h1>
    <p>我是被展开的内容。</p>
    <div data-role="collapsible">
        <h1>单击我-我是嵌套的可折叠块! </h1>
        <p>我是嵌套的可折叠块中被展开的内容。</p>
    </div>
</div>
```

3. 可折叠集合

可折叠集合指被组合在一起的可折叠块。当新块被打开时,所有其他块会关闭。创建若干内容块,然后通过 data-role="collapsible-set",用新容器包装这个可折叠块,如下所示。

```
<div data-role="collapsible-set">
    <div data-role="collapsible">
        <h1>单击我-我可以折叠! </h1>
        <p>我是被展开的内容。</p>
    </div>
    <div data-role="collapsible">
        <h1>单击我-我可以折叠! </h1>
        <p>我是被展开的内容。</p>
    </div>
</div>
```

9.1.8 列表

1. 列表视图

jQuery Mobile 中的列表视图是标准的 HTML 列表:有序列表(<ol>)和无序列表(<ul>)。创建列表,向<ol>或<ul>元素添加 data-role="listview"。如使这些项目可单击,需在每个列表项(<li>)中规定链接。

(1) 常规列表视图

例 9-1-8-1 一个简单的列表视图示例。

① 例 9-1-8-1.htm 源文档。

```
<!DOCTYPE html>
<html>
    <head>
```

```
        <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.mobile
        -1.3.2.min.css">
        <script src="http://code.jquery.com/jquery-1.8.3.min.js"></script>
        <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.js">
       </script>
    </head>
    <body>
        <div data-role="page" id="pageone">
            <div data-role="content">
                <h2>有序列表:</h2>
                <ol data-role="listview">
                    <li><a href="#">列表项</a></li>
                    <li><a href="#">列表项</a></li>
                    <li><a href="#">列表项</a></li>
                </ol>
                <h2>无序列表:</h2>
                <ul data-role="listview">
                    <li><a href="#">列表项</a></li>
                    <li><a href="#">列表项</a></li>
                    <li><a href="#">列表项</a></li>
                </ul>
            </div>
        </div>
    </body>
</html>
```

② Android 平台小米手机浏览器显示结果。

Android 平台小米手机浏览器显示结果如图 9.18 所示。



图 9.18　例 9-1-8-1 的 Android 平台小米手机浏览器显示结果

③ 提示。

如需为列表添加圆角和外边距，可使用 data-inset="true"属性，如下所示。

```
<ul data-role="listview" data-inset="true">
```

默认地，列表中的列表项会自动转换为按钮（无须 data-role="button"）。

（2）列表分隔符

列表分隔符（List Dividers）用于把项目组织和组合为分类/节。如规定列表分隔符，需向<li>元素添加 data-role="list-divider" 属性，如例 9-1-8-2 所示。

例 9-1-8-2 一个具有分隔符的列表视图示例。

① 例 9-1-8-2.htm 源文档。

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.mobile
        -1.3.2.min.css">
        <script src="http://code.jquery.com/jquery-1.8.3.min.js"></script>
        <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.js">
        </script>
    </head>
    <body>
        <div data-role="page" id="pageone">
            <div data-role="content">
                <h2>列表分隔符</h2>
                <ul data-role="listview">
                    <li data-role="list-divider">欧洲</li>
                    <li><a href="#">德国</a></li>
                    <li><a href="#">英国</a></li>
                    <li data-role="list-divider">亚洲</li>
                    <li><a href="#">中国</a></li>
                    <li><a href="#">印度</a></li>
                    <li data-role="list-divider">非洲</li>
                    <li><a href="#">埃及</a></li>
                    <li><a href="#">南非</a></li>
                </ul>
            </div>
        </div>
    </body>
</html>
```

② Android 平台小米手机浏览器显示结果。

Android 平台小米手机浏览器显示结果如图 9.19 所示。

图 9.19　例 9-1-8-2 的 Android 平台小米手机浏览器显示结果

③ 提示。

如果列表是字母顺序的，jQuery Mobile 自动添加恰当的分隔符，通过在<ol>或<ul>元素上设置 data-autodividers＝"true"属性实现。data-autodividers＝"true"属性通过对列表项文本的首字母进行大写来创建分隔符。

（3）搜索过滤器

如在列表中添加搜索框，需使用 data-filter＝"true"属性，如例 9-1-8-3 所示。

例 9-1-8-3　一个具有搜索过滤器的列表视图示例。

① 例 9-1-8-3. htm 源文档。

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.mobile
        -1.3.2.min.css">
        <script src="http://code.jquery.com/jquery-1.8.3.min.js"></script>
        <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.js">
        </script>
    </head>
    <body>
        <div data-role="page" id="pageone">
            <div data-role="content">
                <h2>我的通讯录</h2>
                <ul data-role="listview" data-autodividers="true" data-inset="
                true" data-filter="true">
                    <li><a href="#">Adele</a></li>
                    <li><a href="#">Agnes</a></li>
```

```
                    <li><a href="#">Albert</a></li>
                    <li><a href="#">Billy</a></li>
                    <li><a href="#">Bob</a></li>
                    <li><a href="#">Calvin</a></li>
                    <li><a href="#">Cameron</a></li>
                    <li><a href="#">Chloe</a></li>
                    <li><a href="#">Christina</a></li>
                    <li><a href="#">Diana</a></li>
                    <li><a href="#">Gabriel</a></li>
                    <li><a href="#">Glen</a></li>
                    <li><a href="#">Ralph</a></li>
                    <li><a href="#">Valarie</a></li>
                </ul>
            </div>
        </div>
    </body>
</html>
<ul data-role="listview" data-filter="true"></ul>
```

② Android 平台小米手机浏览器显示结果。

Android 平台小米手机浏览器显示结果如图 9.20 所示。

③ 提示。

默认地，搜索框中的文本是 Filter items...。如需修改默认文本，使用 data-filter-placeholder 属性，如下所示。

```
<ul data-role="listview" data-filter="true" data-filter-placeholder="搜索姓名">
```

2. 列表内容

(1) 列表缩略图

对于分辨率大于 16×16 的图像，在链接中添加<img>元素，jQuery Mobile 将自动把图像调整至 80×80。

例 9-1-8-4 一个简单的列表缩略图示例。

① 例 9-1-8-4.htm 源文档。

```
<!DOCTYPE html>
<html>
  <head>
        <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.mobile
        -1.3.2.min.css">
        <script src="http://code.jquery.com/jquery-1.8.3.min.js"></script>
        <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.js">
       </script>
    </head>
```



图 9.20　例 9-1-8-3 的 Android 平台小米手机浏览器显示(搜索框未输入内容)结果

```
<body>
    <div data-role="page" id="pageone">
        <div data-role="content">
            <h2>包含缩略图和文本的列表:</h2>
            <ul data-role="listview" data-inset="true">
                <li>
                    <a href="#">
                        <img src="/Images/chrome.png">
                        <h2>Google Chrome</h2>
```

```
                                <p>Google Chrome is a free, open-source web
                                browser. Released in 2008.</p>
                            </a>
                        </li>
                        <li>
                            <a href="#">
                                <img src="/Images/firefox.png">
                                <h2>Mozilla Firefox</h2>
                                <p>Firefox is a web browser from Mozilla. Released
                                in 2004.</p>
                            </a>
                        </li>
                    </ul>
                </div>
            </div>
        </body>
</html>
```

② Android 平台小米手机浏览器显示结果。

Android 平台小米手机浏览器显示结果如图 9.21 所示。

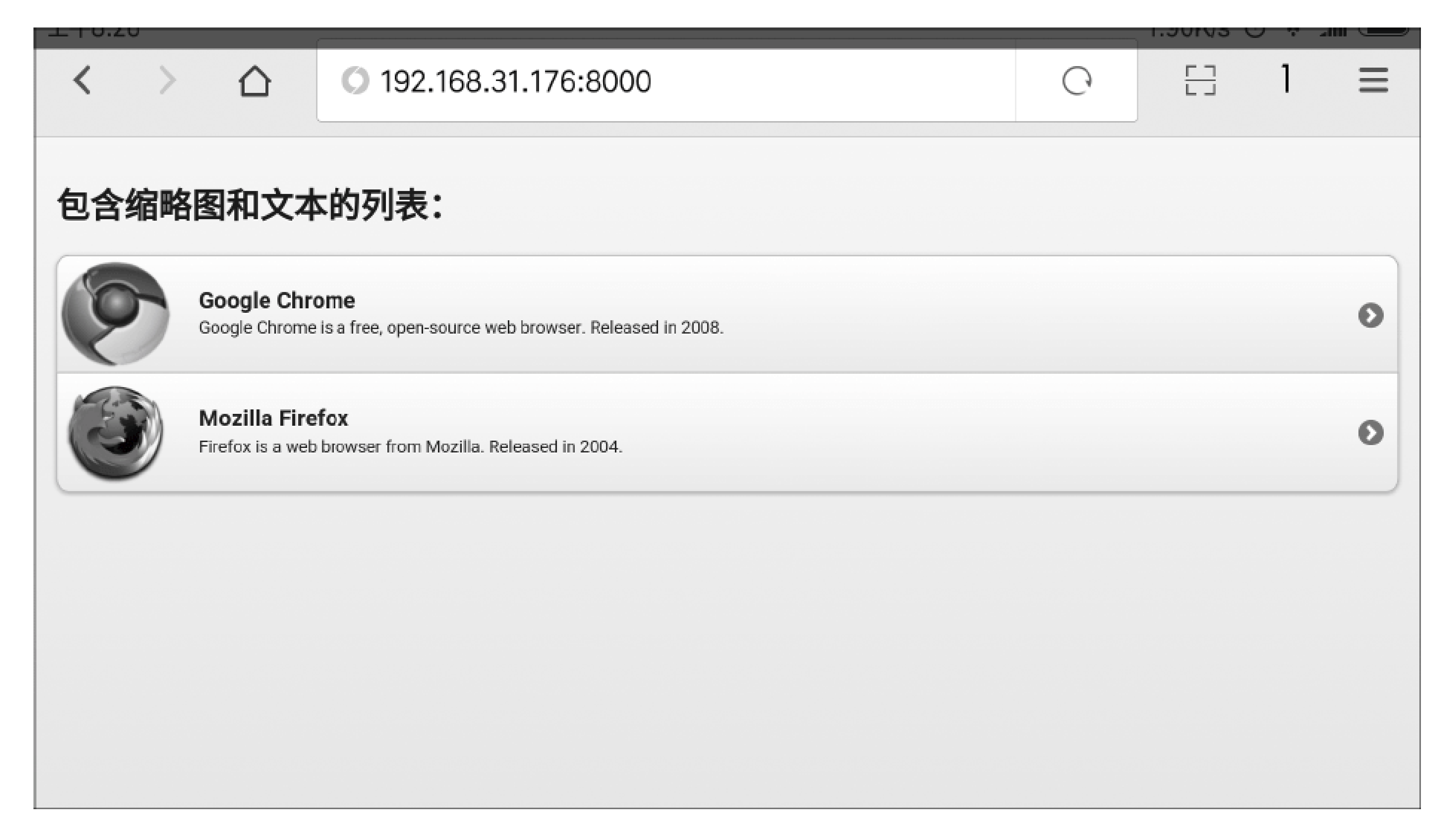


图 9.21　例 9-1-8-4 的 Android 平台小米手机浏览器显示结果

③ 提示。

如向列表添加分辨率为 16×16 的图标，需向<img>元素添加 class="ui-li-icon"属性，如下所示。

```
<li><a href="#"><img src="us.png" alt="USA" class="ui-li-icon">USA</a>
</li>
```

(2) 拆分按钮

如创建带有垂直分隔栏的拆分列表，需在<li>元素内放置两个链接。jQuery Mobile 会自动为第 2 个链接添加蓝色箭头图标的样式，链接中的文本(如有)将在用户划过该图标时显示，如例 9-1-8-5 所示。

例 9-1-8-5 具有拆分按钮的列表

① 例 9-1-8-5.htm 直接相关的源文档。

```
<div data-role="content">
      <h2>拆分按钮</h2>
      <ul data-role="listview" data-inset="true">
            <li>
                  <a href="#"><img src="/Images/chrome.png"><h2>Google Chrome</h2>
                        <p>Google Chrome is a free, open-source web browser. Released in
                        2008.</p>
                  </a>
                  <a href="#">Some Text</a>
              </li>
              <li>
                  <a href="#"><img src="/Images/firefox.png"><h2>Mozilla Firefox</
                  h2>
                        <p>Firefox is a web browser from Mozilla. Released in 2004.</p>
                  </a>
                  <a href="#">Some Text</a>
              </li>
      </ul>
</div>
```

② Android 平台小米手机浏览器显示结果。

Android 平台小米手机浏览器显示结果如图 9.22 所示。

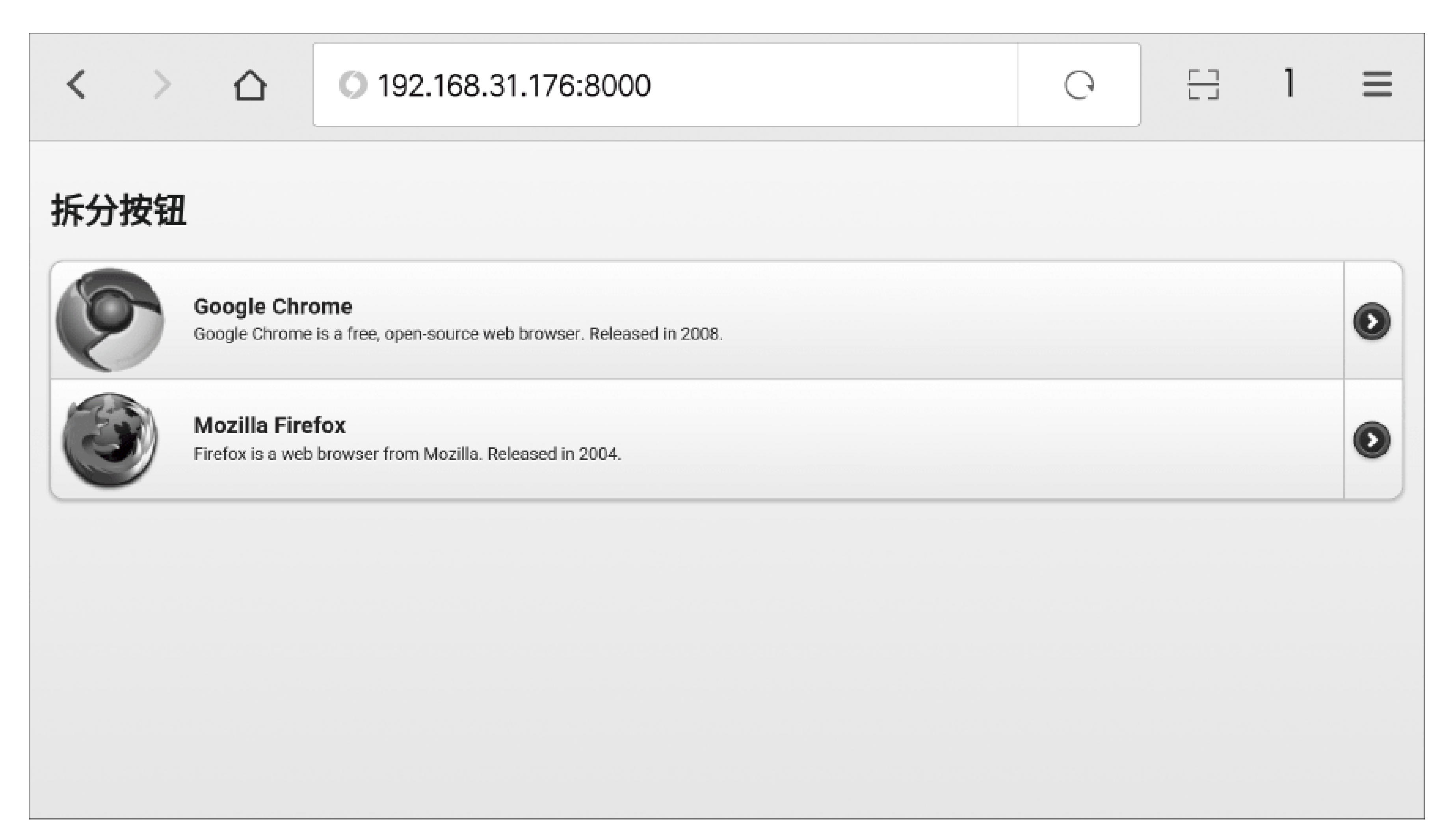


图 9.22 例 9-1-8-5 的 Android 平台小米手机浏览器显示结果

(3) 计数泡沫

计数泡沫用于显示与列表项相关的数目。如添加计数泡沫，需使用行内元素，比如<span>，设置 class 含有 ui-li-count 属性并添加数字，如例 9-1-8-6 所示。

例 9-1-8-6 分类显示邮箱邮件数。

① 例 9-1-8-6.htm 源文档。

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.mobile
        -1.3.2.min.css">
        <script src="http://code.jquery.com/jquery-1.8.3.min.js"></script>
        <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.js">
        </script>
    </head>
    <body>
        <div data-role="page" id="pageone">
            <div data-role="content">
                <h2>我的邮箱</h2>
                <ul data-role="listview" data-inset="true">
                    <li><a href="#">收件箱<span class="ui-li-count">25</
                    span></a></li>
                    <li><a href="#">发件箱<span class="ui-li-count">432</
                    span></a></li>
                    <li><a href="#">垃圾箱<span class="ui-li-count">7</span
                    ></a></li>
                </ul>
            </div>
        </div>
    </body>
</html>
```

② Android 平台小米手机浏览器显示结果。

Android 平台小米手机浏览器显示结果如图 9.23 所示。

③ 提示。

如在计数泡泡中显示正确的数字,必须实时更新。

9.1.9 事件

jQuery Mobile 不但可以使用任何标准的 jQuery 事件,还可使用为移动浏览定制的事件,具体如下。

① 触摸事件:当用户触摸屏幕时触发(敲击和滑动)。

② 滚动事件:当上下滚动时触发。

③ 方向事件:当设备垂直或水平旋转时触发。

④ 页面事件:当页面被显示、隐藏、创建、加载以及/或卸载时触发。

上述 4 种移动事件均由文档 on 方法来关联,由第一个实参表示不同事件。

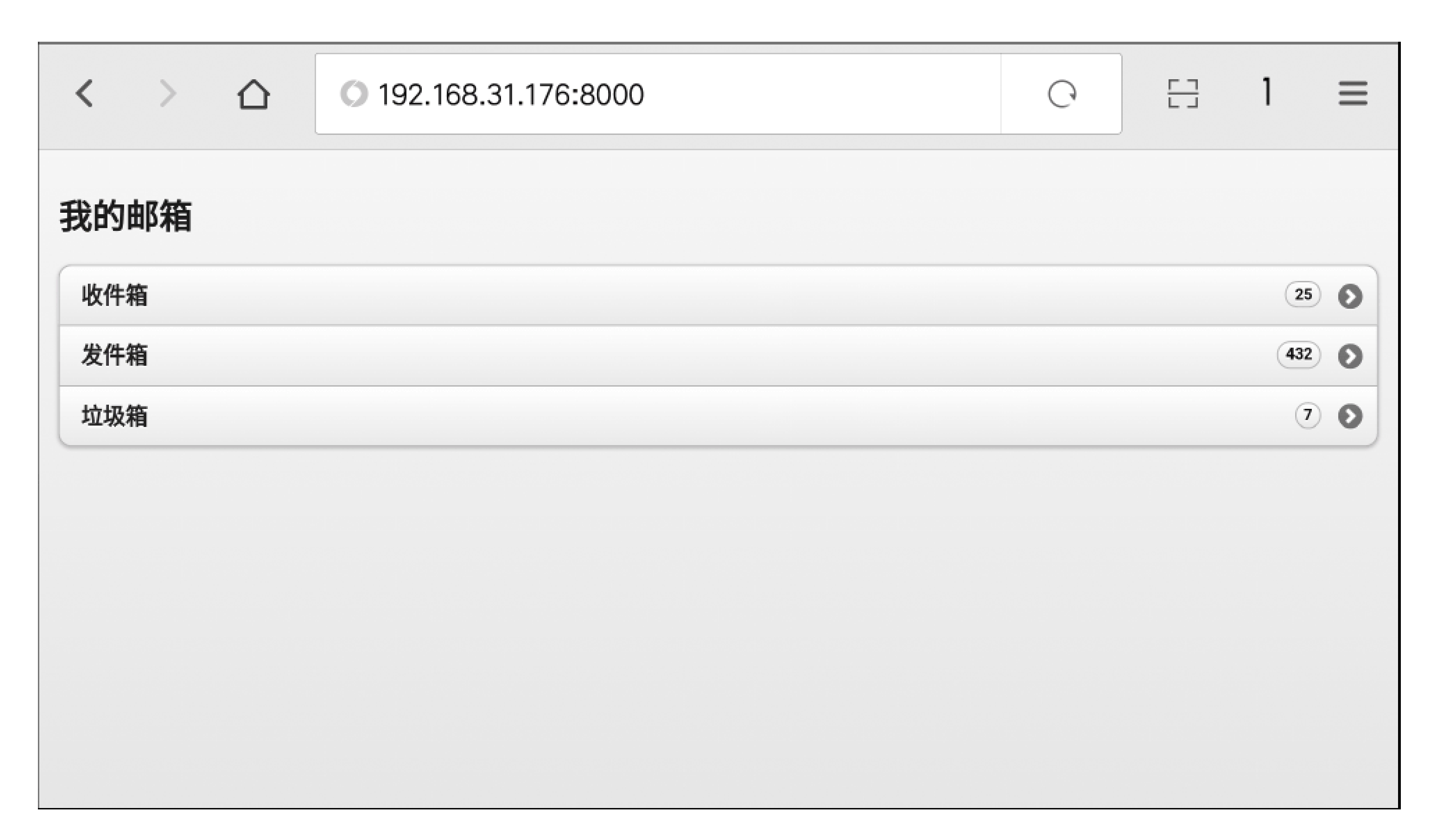


图 9.23　例 9-1-8-6 的 Android 平台小米手机浏览器显示结果

1. 初始化事件

在 jQuery 中已学到使用文档 ready 事件来阻止 jQuery 代码在文档结束加载前运行。然而，在 jQuery Mobile 中使用 pageinit 事件，它在页面已初始化并完善样式设置后发生。例如：

```
<script>
    $(document).on("pageinit","#pageone",function(){
        //此处是 jQuery 事件...
    });
</script>
```

文档 on 方法的第一个实参 pageinit 表示初始化事件，第二个实参 # pageone 表示指向页面的 id。

2. 触摸事件

触摸事件在用户触摸屏幕（页面）时触发，等同于 PC 桌面的单击事件。

（1）tap 事件

tap 事件在敲击某个元素时触发。如当单击 <p> 元素时，隐藏当前 <p> 元素，语句如下。

```
$("p").on("tap",function(){ $(this).hide(); });
```

（2）taphold 事件

taphold 事件在敲击某个元素并保持 1 秒时被触发，如下所示。

```
$("p").on("taphold",function(){ $(this).hide(); });
```

(3) swip 事件

swipe 事件是在某个元素上水平滑动超过 30px 时被触发，如下所示。

```
$("p").on("swipe",function(){ $("span").text("Swipe detected!"); });
```

(4) swipleft 事件

swipeleft 事件是在某个元素上从左滑动超过 30px 时被触发，如下所示。

```
$("p").on("swipeleft",function(){ alert("You swiped left!"); });
```

(5) swiperight 事件

swiperight 事件是在某个元素上从右滑动超过 30px 时被触发，如下所示。

```
$("p").on("swiperight",function(){ alert("You swiped right!"); });
```

3. 滚动事件

jQuery Mobile 提供滚动开始和当滚动结束 2 个滚动事件。

(1) scrollstart 事件

scrollstart 事件在用户开始滚动页面时被触发，如下所示。

```
$(document).on("scrollstart",function(){ alert("开始滚动!"); });
```

iOS 设备会在滚动事件发生时冻结 DOM 操作，这意味着当用户滚动时无法改变任何事物。

(2) scrollstop 事件

scrollstop 事件在用户停止滚动页面时被触发，如下所示。

```
$(document).on("scrollstop",function(){ alert("结束滚动!"); });
```

4. 方向事件

方向事件在用户垂直或水平旋转移动设备时被触发。如使用 orientationchange 事件，需把它添加到 window 对象，如下所示。

```
$(window).on("orientationchange", function(){ alert("方向已改变!"); });
```

callback 函数可以设置一个参数，即 event 对象，它会返回移动设备的方向，即 portrait(垂直)或 landscape(水平)，如下所示。

```
$(window).on("orientationchange",function(event){ alert("方向是: " + event.
orientation);});
```

由于 orientationchange 事件与 window 对象绑定，可使用 window.orientation 属性来判断方向，不同方向设置不同样式，以区分 portrait 和 landscape 视图，如下所示。

```
$(window).on("orientationchange",function(){
    if(window.orientation==0) //Portrait
    {
```

```
        $("p").css({"background-color":"yellow","font-size":"300%"});
    }
    else //Landscape
    {
        $("p").css({"background-color":"pink","font-size":"200%"});
    }
});
```

5. 页面事件

jQuery Mobile 与页面打交道的事件分如下 4 类。

① 页面初始化：在页面创建前、页面创建时以及页面初始化之后。

② 页面加载/卸载：当外部页面加载时、卸载时或遭遇失败时。

③ 页面过渡：在页面过渡之前和之后。

④ 页面改变：当页面被更改或遭遇失败时。

(1) 初始化事件

jQuery Mobile 初始化一张典型页面时，经历 3 个阶段，分别是页面创建前、页面创建时和页面初始化后，具体如下。

① pagebeforecreate 事件。

当页面即将初始化，并且在 jQuery Mobile 已开始增强页面之前，触发该事件。

② pagecreate 事件。

当页面已创建，但增强完成之前，触发该事件。

③ pageinit 事件。

当页面已初始化，并且在 jQuery Mobile 已完成页面增强之后，触发该事件。

每个阶段触发的事件都可用于插入或操作代码。下面的例子演示在 jQuery Mobile 中创建页面时何时触发每种事件。

```
$(document).on("pagebeforecreate",function(event){
    alert("触发 pagebeforecreate 事件!");
});
$(document).on("pagecreate",function(event){
    alert("触发 pagecreate 事件!");
});
$(document).on("pageinit",function(event){
    alert("触发 pageinit 事件!")
});
```

(2) 加载事件

页面加载事件属于外部页面。外部页面载入 DOM，将触发两个事件：pagebeforeload、pageload（成功）或 pageloadfailed（失败），具体描述如下。

① pagebeforeload：在任何页面加载请求做出之前触发。

② pageload：在页面已成功加载并插入 DOM 后触发。

③ pageloadfailed：如果页面加载请求失败，则触发该事件，默认显示 Error Loading

Page 消息。

下面的例子演示 pageload 和 pagloadfailed 事件的工作原理。

```
$(document).on("pageload", function(event, data){
    alert("触发 pageload 事件! \nURL: "+data.url);
});
$(document).on("pageloadfailed", function(event, data){
    alert("抱歉,被请求页面不存在。");
});
```

(3) 过渡事件

从一页过渡到下一页时产生页面过渡事件。页面过渡涉及两个页面：一张“来”的页面和一张“去”的页面。过渡使当前活动页面到新页面的改变过程变得更加动感。具体过渡事件及其描述如下。

① pagebeforeshow：在“去的”页面触发，在过渡动画开始前。

② pageshow：在“去的”页面触发，在过渡动画完成后。

③ pagebeforehide：在“来的”页面触发，在过渡动画开始前。

④ pagehide：在“来的”页面触发，在过渡动画完成后。

下面的例子演示了过渡事件的工作原理。

```
$(document).on("pagebeforeshow","#pagetwo",function(){ //当进入页面二时
    alert("页面二即将显示");
});
$(document).on("pageshow","#pagetwo",function(){ //当进入页面二时
    alert("现在显示页面二");
});
$(document).on("pagebeforehide","#pagetwo",function(){ //当离开页面二时
    alert("页面二即将隐藏");
});
$(document).on("pagehide","#pagetwo",function(){ //当离开页面二时
    alert("现在隐藏页面二");
});
```

9.2 上机实验样例

本章所有上机实验需要具备无线上网环境，以使移动设备能够访问 IIS 服务器发布 Web 站点。本章源代码需要发布到一个 Web 站点，移动设备才能使用浏览器访问。本章实验分为安装 IIS、发布 Web 站点、设置防火墙和访问浏览四大步骤。

9.2.1 安装 IIS

下面以 Windows 8 专业版为例讲解安装过程。

① 选择控制面板的"程序"。

单击屏幕左下角的开始菜单按钮,进入控制面板,如图 9.24 所示。在图 9.24 中选择"程序",进入图 9.25 所示的界面。



图 9.24　Windows 8.0 企业版的控制面板



图 9.25　控制面板的程序选项

② 启用 Internet 信息服务和 Internet Information Services 可承载的 Web 核心。

在图 9.25 中选择"启用或关闭 Windows 功能",进入 Windows 功能对话框,选中 Internet 信息服务和 Internet Information Services 可承载的 Web 核心,如图 9.26 所示。

单击图 9.26 中的"确定"按钮,系统开始自动安装 IIS 的相关应用。安装完毕单击屏幕左下角的"开始"菜单图标,展开"所有程序"后单击 IIS 文件夹,得到如图 9.27 所示的信息服务 IIS 管理器,表示安装 IIS 成功。

9.2.2　发布 Web 站点

① 启动 Internet 信息服务(IIS)管理器。



图 9.26 Windows 功能选项卡



图 9.27 Internet 信息服务(IIS)管理

单击图 9.28 的“Internet 信息服务(IIS)管理器”选项，则启动 Internet 信息服务(IIS)管理器，进入图 9.27 所示的对话框。

② 添加网站。

展开计算机-LHB(lhb/lhb_dell)，选中网站，右击，弹出快捷菜单，如图 9.29 所示。选择“添加网站”选项，弹出如图 9.30 所示的对话框。各项参数如图 9.30 所示，单击“确定”按钮，则成功添加网站。

③ 查看应用缓冲池参数。

单击图 9.29 所示的“应用程序池”选项，进入图 9.31 所示的对话框，确认 jQueryMobile 的.NET Framework 版本为 v4.0。如不是 v4.0，双击 jQueryMobile，可进一步编辑应用程序缓冲池，修改其中的版本。

④ 启动目录浏览。



图 9.28 安装 IIS 后的开始菜单



图 9.29 在快捷菜单中选择“添加网站”选项

选中 jQueryMobile 网站，单击“目录浏览”选项，如果管理网站的启动已经处于灰色状态，表示网站目录浏览已经启用。否则单击“启动”选项，如图 9.32 所示。

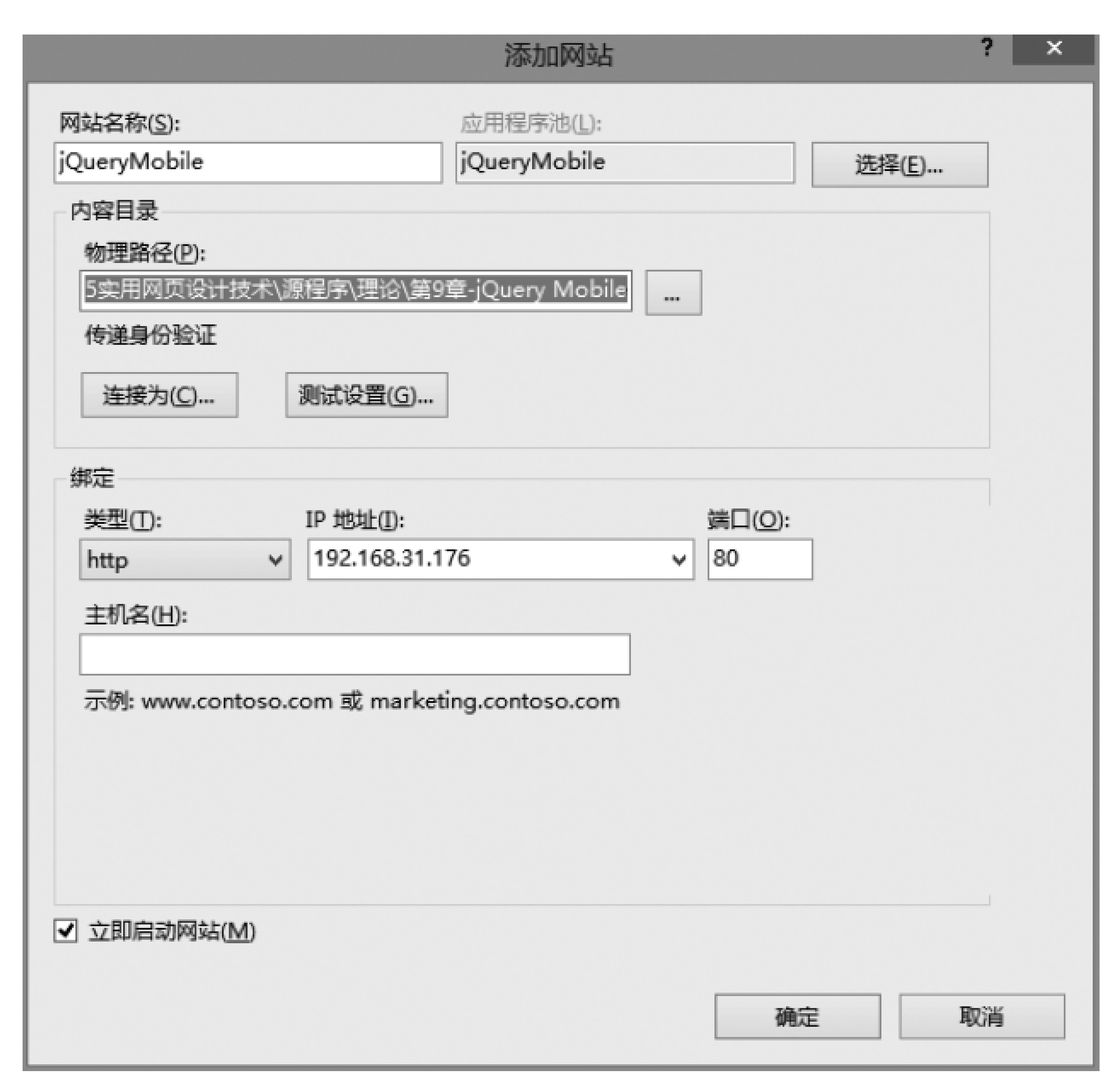


图 9.30 “添加网站”对话框



图 9.31 查看 jQueryMobile 应用缓冲池



图 9.32 操作目录浏览选项

9.2.3 设置 http 访问通过防火墙

① 启动控制面板,如图 9.33 所示。



图 9.33 控制面板

② 选择"网络和 Internet"选项,打开"网络和 Internet"对话框,如图 9.34 所示。



图 9.34 "网络和 Internet"对话框

③ 启动网络和共享中心,如图 9.35 所示。

④ 选择"Windows 防火墙"选项,打开"Windows 防火墙"对话框,如图 9.36 所示。

⑤ 单击"允许应用或功能通过 Windows 防火墙",打开"允许的应用"对话框,如图 9.37 所示。

⑥ 如果不能选中"万维网服务"的专用和公用 2 个复选框,可单击"更改设置"按钮,而后选中。否则,直接选中"万维网服务"的专用和公用 2 个复选框,如图 9.38 所示。

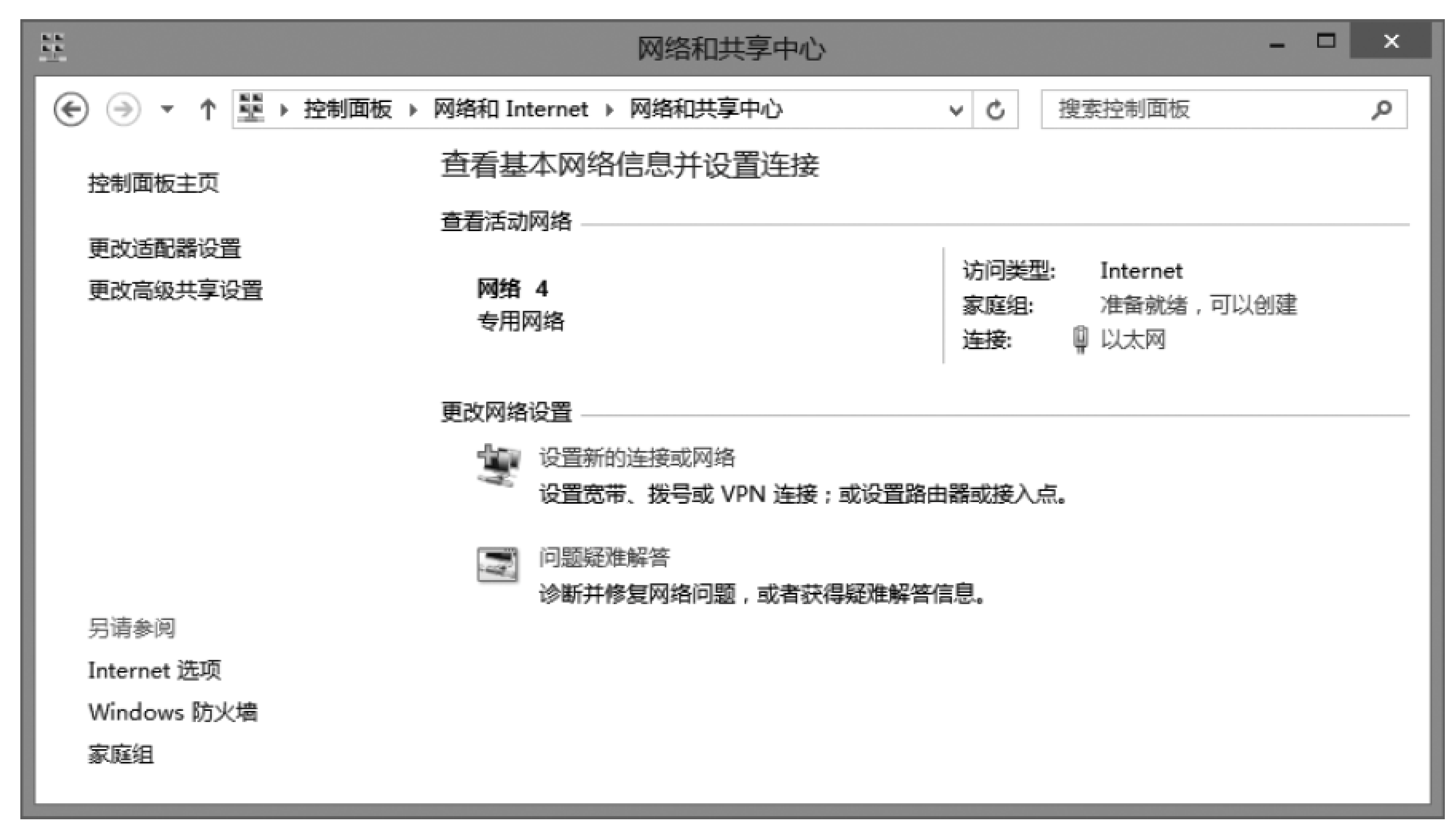


图 9.35 “网络和共享中心”对话框

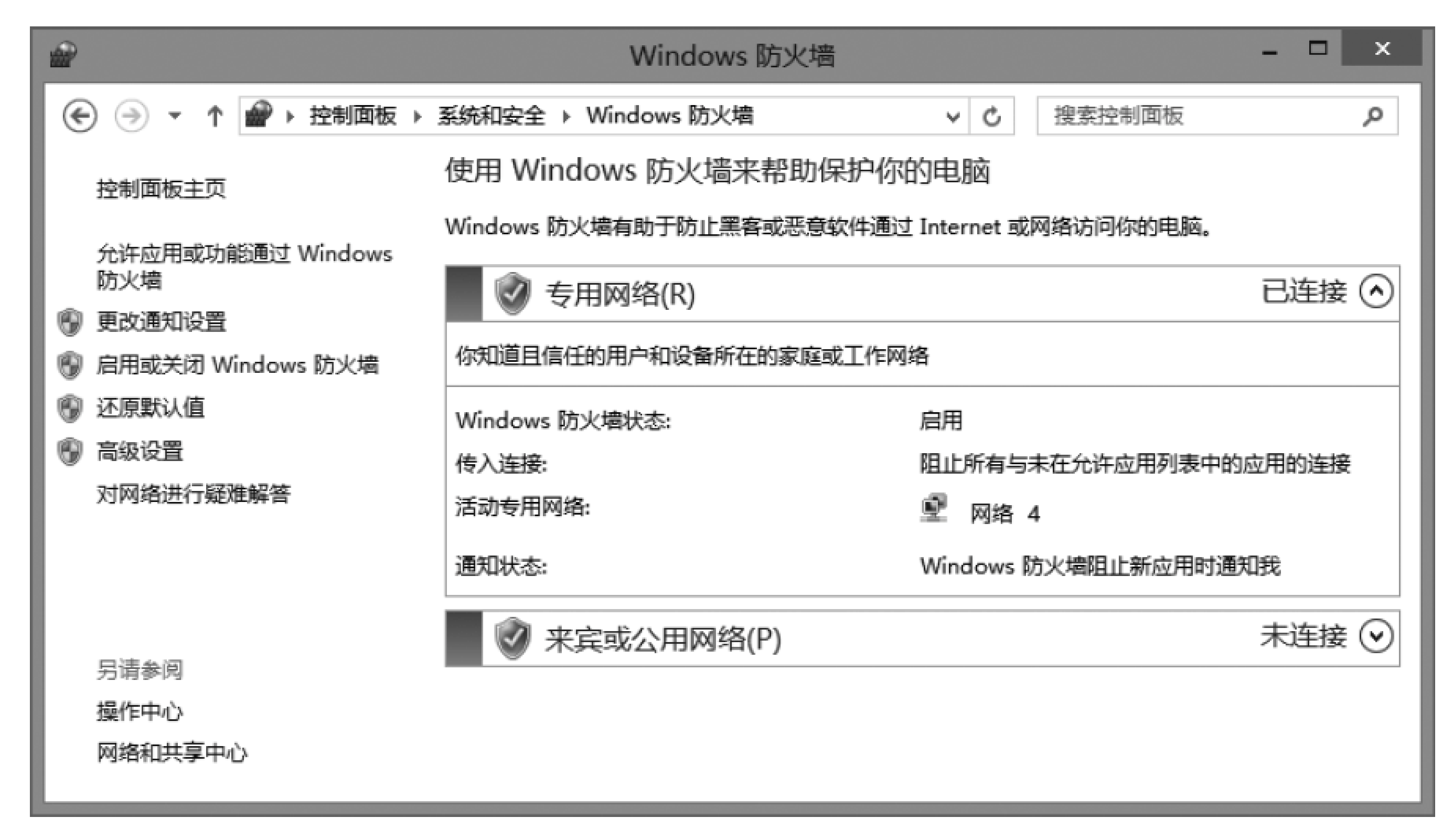


图 9.36 “Windows 防火墙”对话框



图 9.37 “允许的应用”对话框

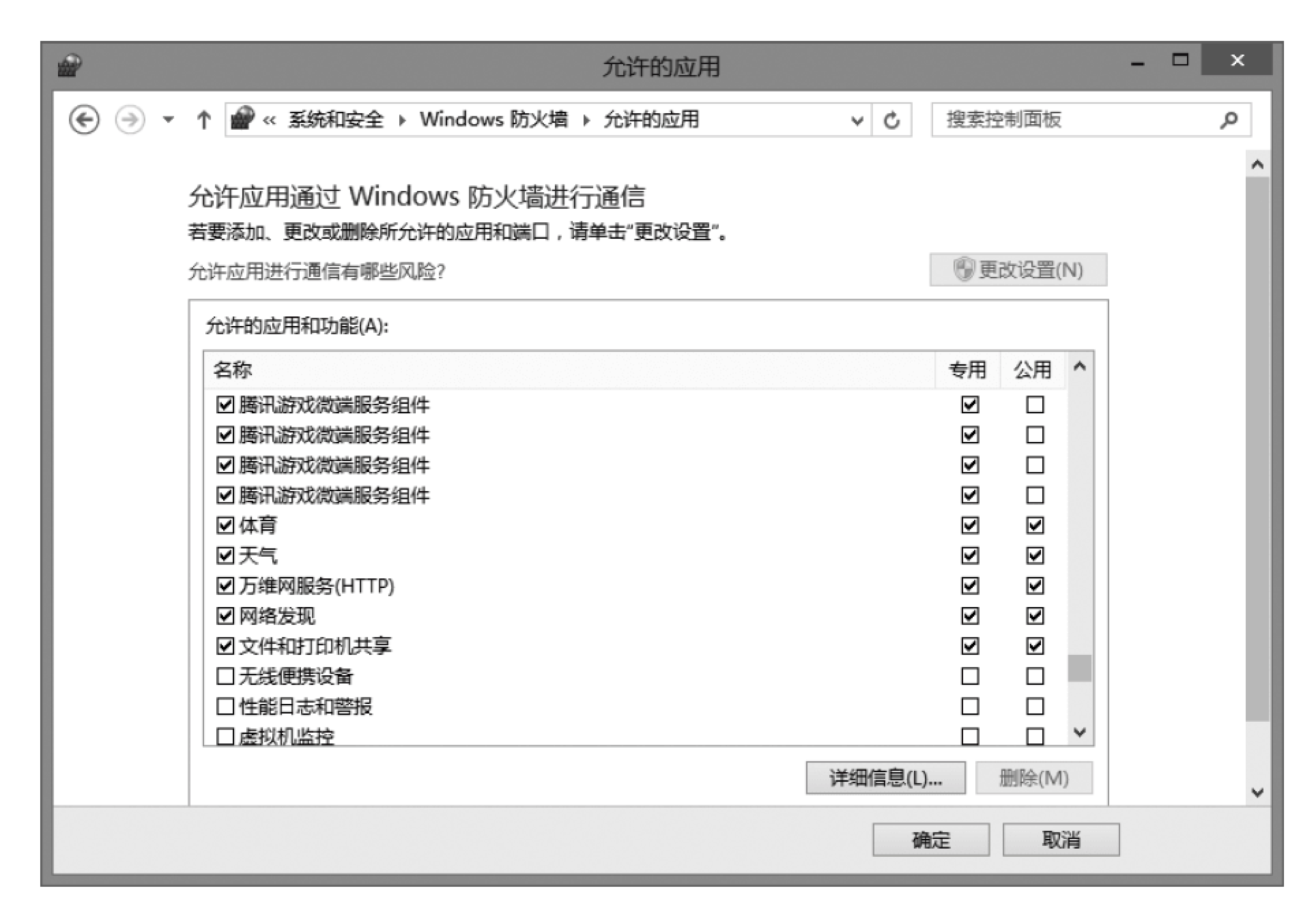


图 9.38　在防火墙允许的应用列表中选中万维网服务(HTTP)的专用和公用选项

⑦ 单击图 9.38 所示的"确定"按钮，结束设置。

9.2.4　组建无线局域网

在 Windows 10 操作系统平台下，以 D-LINK 无线路由器实物连接配置为例，讲述无线局域网的配置。首先将计算机连接到无线路由器 LAN 口访问配置网页，而后进行配置，最后将无线路由器连入 WAN，具体如下。

1. 计算机访问无线路由器配置网页

把 D-LINK 无线路由器转过来，查看背部的路由器 IP 地址、用户名和密码信息。根据 D-LINK 无线路由器的 IP 地址 192.168.0.1，配置 D-LINK 无线路由器的笔记本电脑的 IP 地址为 192.168.0.2，使笔记本电脑能够访问无线路由器内的配置页面。步骤如下。

① 连接笔记本电脑和无线路由器。用带有 RJ45 头(水晶头)网线的一端连到无线路由器的 LAN(图中的 4 个黑色端口之一)，另一端连到笔记本电脑的网口，如图 9.39 所示。

② 设置笔记本电脑的 IP 地址。右击桌面右下角任务栏的网络图标，弹出如图 9.40 所示的快捷菜单。选择"打开网络和共享中心"选项，打开如图 9.41 所示的"网络和共享中心"对话框。单击"以太网"选项，打开如图 9.42 所示的"以太网状态"对话框。单击"属性"按钮，打开如图 9.43 所示的"以太网属性"对话框。双击"Internet 协议版本 4(TCP/IPv4)"选项，打开如图 9.44 所示的"Internet 协议版本 4(TCP/IPv4)属性"对话框，输入如图 9.44 所示的 IP 地址和子网掩码，单击"确定"按钮。



图 9.39　将 D-Link 路由器连接到配置计算机



图 9.40　网络任务快捷菜单



图 9.41　“网络和共享中心”对话框



图 9.42 “以太网状态”对话框



图 9.43 “以太网属性”对话框

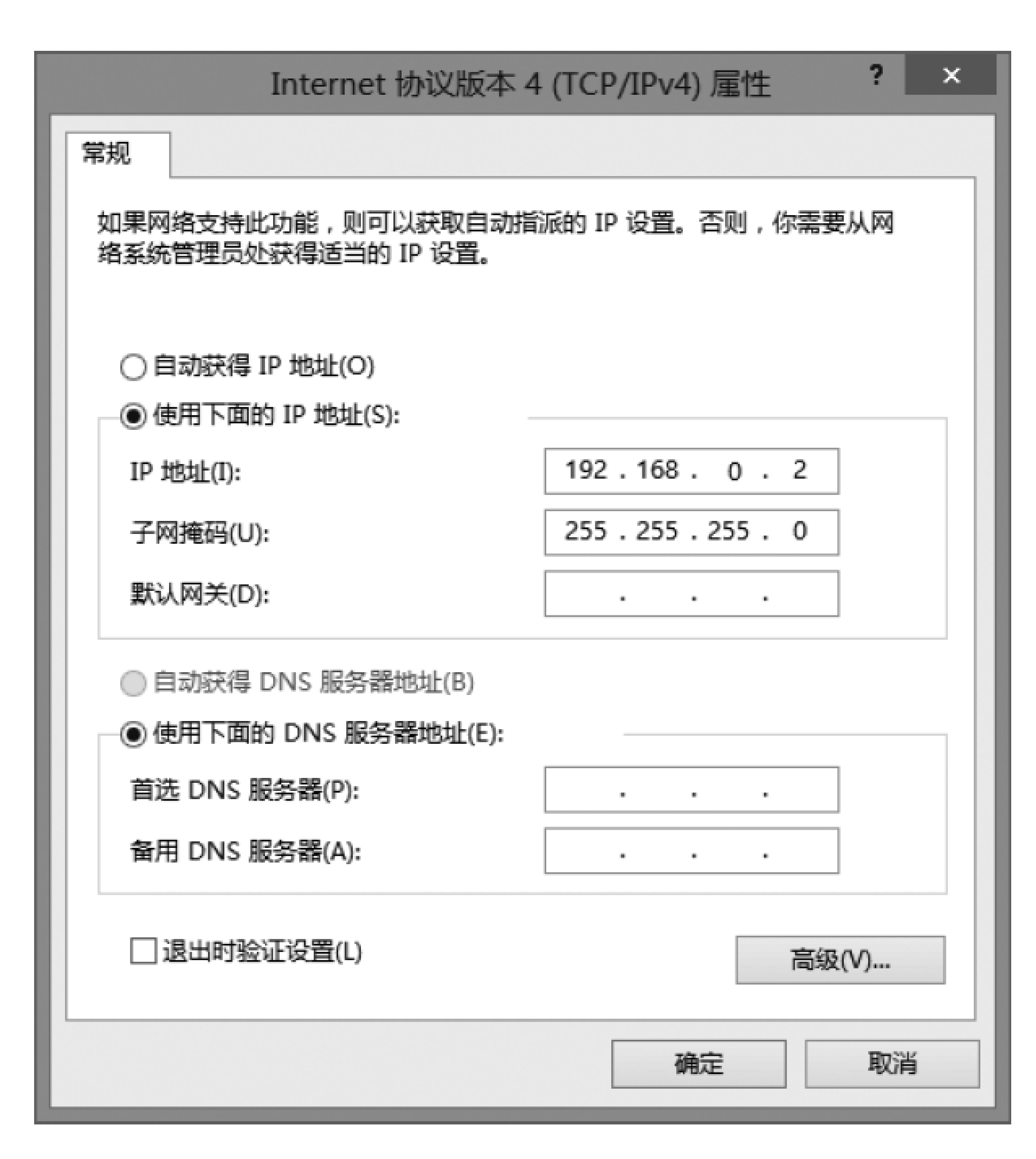


图 9.44 “Internet 协议版本 4(TCP/IPv4)属性”对话框

③ 打开一个浏览器，在地址栏中输入 192.168.0.1，按回车键进入路由器配置页面，如图 9.45 所示。



图 9.45 D-Link 登录页面

2. 配置无线路由器

在图 9.45 中输入密码，单击“确定”按钮。成功登录后，进入如图 9.46 所示的 D-Link 主界面。采用 INTERNET 连接设置向导的方式进行设置，步骤如下。

① 选中顶部主菜单的“设置”。

② 选中左侧菜单的 Internet 设置。

③ 单击图 9.46 中部的“Internet 连接设置向导”按钮，进入如图 9.47 所示的对话框。



图 9.46 D-Link 主界面



图 9.47 "Internet 连接设置向导"对话框

④ 单击图 9.47 所示对话框中的"下一步"按钮，进入如图 9.48 所示的对话框。

⑤ 设置好密码，单击图 9.48 所示的"下一步"按钮，进入如图 9.49 所示的对话框。

⑥ 设置好时区，单击图 9.49 所示的"下一步"按钮，进入如图 9.50 所示的对话框。

⑦ 选择静态 IP 地址连接方式，单击图 9.50 所示的"下一步"按钮，进入如图 9.51 所示的对话框。

⑧ 在图 9.51 中设置好 IP 地址、子网掩码、网关和 DNS，单击"下一步"按钮，进入如图 9.52 所示的对话框。注意，LAN 网和 WAN 网不能处于同一网段（同一网段指 IP 地址和子网掩码相与得到相同的网络地址）。这里 LAN 网的网段为 192.168.0，而 WAN 网段为 202.194.52，二者不同。当网段相同时，只能修改 LAN 网的 IP 地址或子网掩码。WAN 网的网段由学校的网络中心规划。



图 9.48 “设置账户密码”对话框



图 9.49 设置时区对话框



图 9.50 配置 Internet 连接对话框



图 9.51 设置静态 IP 地址连接对话框



图 9.52 配置完成进行连接对话框

3. 连接 WAN 口到交换机

一般路由器都有几个网线接口，分为两种颜色，有一个口的颜色和其他口的不同，这就是 WAN 口，图 9.53 所示的下方的黄色端口为 WAN 口，4 个黑色端口为 LAN 口。用一根 2 米左右、两头有水晶头的网线的一端连到无线路由器的 WAN 口，如图 9.53 所示。另一端插到能上网的端口上，如图 9.54 所示。图 9.54 的端口通过地下布连线连到图 9.55 所示的交换机上。

物理连接完成后，单击图 9.51 所示的连接按钮，则连接成功。



图 9.53 网线的一端连到路由器黄色端口



图 9.54 网线的另一端连到能上网的端口



图 9.55 网线另一端的端口已连到交换机

9.2.5 手机中浏览网页

实验用手机为小米手机,MIUI 版本为 MIUI 7.4.13 开发版。假定浏览例 9-1-2-3.htm,首先要选择 WLAN 连接,而后设置例 9-1-2-3.htm 为第一个默认文档,最后在手机浏览器中浏览。

1. 选择 WLAN 连接

在手机桌面上选择“设置”,单击 WLAN,进入 WLAN 操作界面。开启 WLAN,选择 D-Link_DIR-820LW-5GHz 连接,连接密码为 123456abc。连接成功后,WLAN 界面如图 9.56 所示。

2. 设置例 9-1-2-3.htm 为第一个默认文档

如果例 9-1-2-3.htm 不在当前默认文档中,可按如下步骤设置。

① 选中网站 jQueryMobile。

② 双击默认文档选项。

③ 在默认文档空白区域中右击,弹出快捷菜单,如图 9.57 所示。



图 9.56 WLAN 操作界面



图 9.57　在 jQueryMobile 网站的默认文档中右击激活快捷菜单

④ 单击"添加"按钮,弹出"添加默认文档"对话框。

⑤ 在对话框中输入"例 9-1-2-3.htm",如图 9.58 所示。

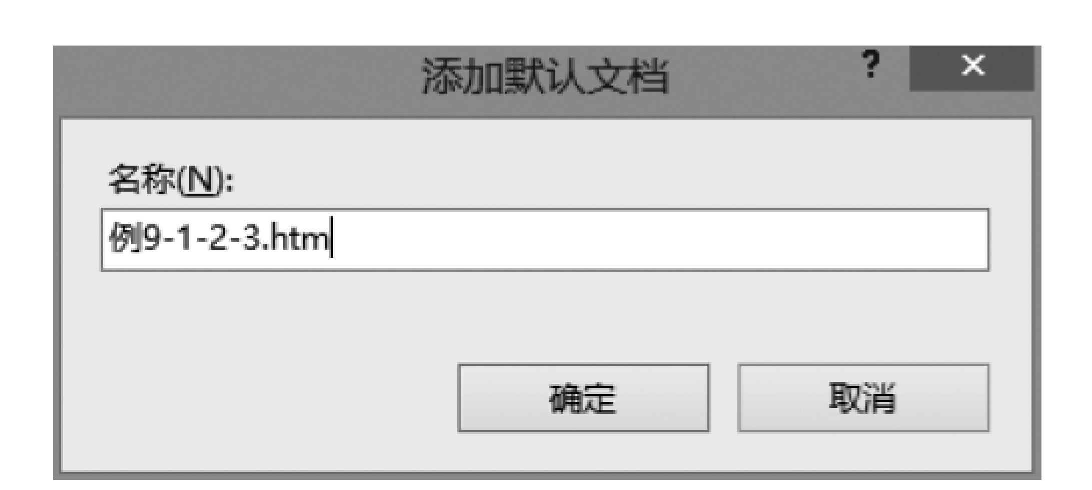


图 9.58　在"添加默认文档"对话框中输入文档名称

⑥ 单击"确定"按钮,设置成功后如图 9.59 所示。

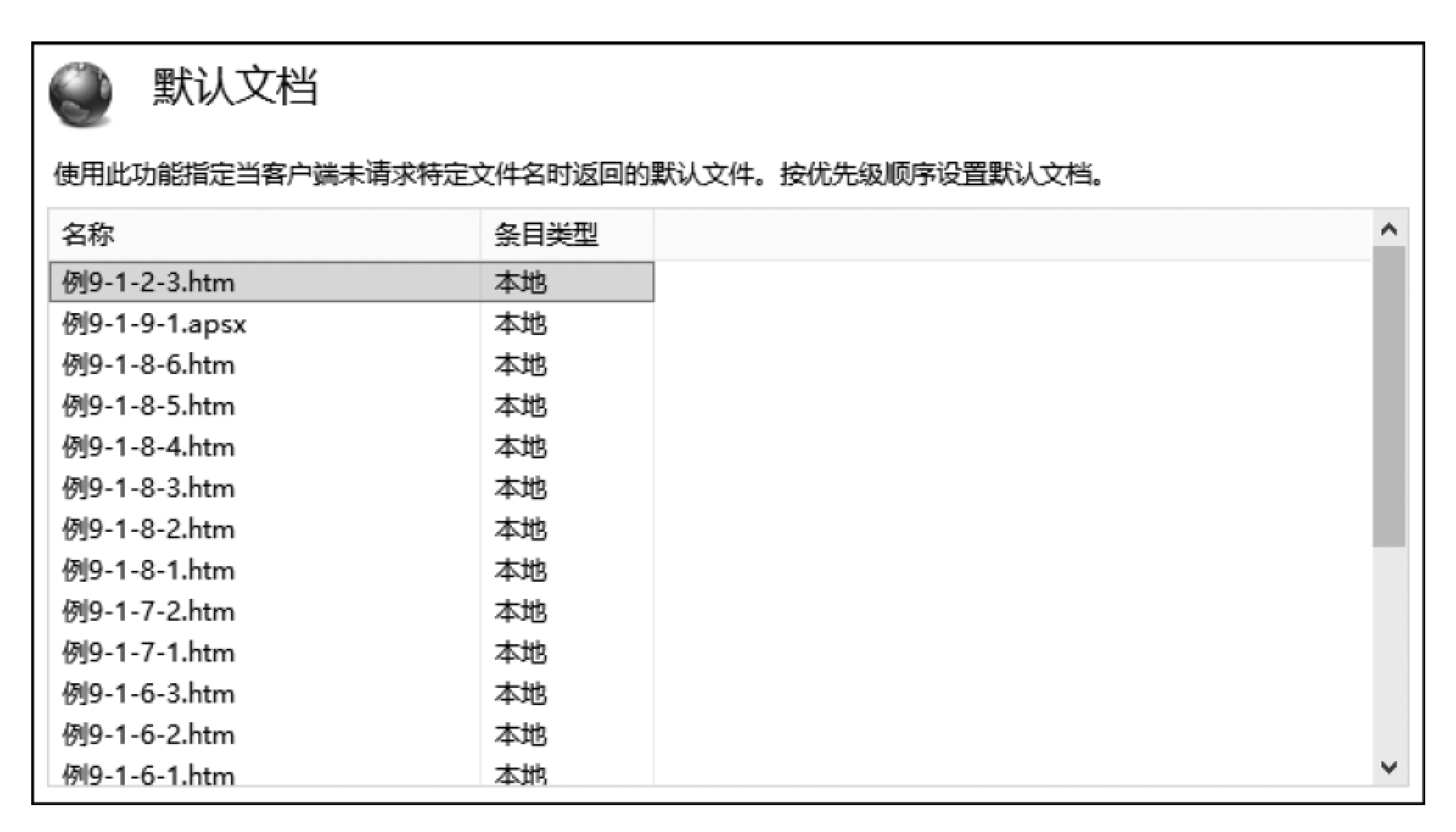


图 9.59　成功添加例 9-1-2-3.htm 为默认文档后的默认文档列表

假定浏览例 9-1-2-3.htm，而且例 9-1-2-3.htm 已在当前默认文档中，但不是第一个文档，按如下步骤设置。

① 选中网站 jQueryMobile。

② 双击默认文档选项。

③ 选中默认文档例 9-1-2-3.htm，右击，弹出快捷菜单，如图 9.60 所示。

④ 单击上移。

⑤ 重复步骤③和④，直到例 9-1-2-3.htm 移至第一个。

添加...
删除
上移
下移
禁用
恢复为父项
帮助
联机帮助

图 9.60 默认文档操作菜单

3. 在手机浏览器中浏览

① 打开手机，启动浏览器。

② 在浏览器的地址栏中输入 http://192.168.31.176。

③ 单击地址栏右侧(或软键盘右下方)的“前往”按钮，得到如图 9.61 所示的网页。

说明：如果地址栏中未输入端口，表示使用默认端口 80。否则必须含有端口信息。如添加网站时设置端口为 8000，则浏览器地址栏中应输入 http://192.168.31.176:8000。

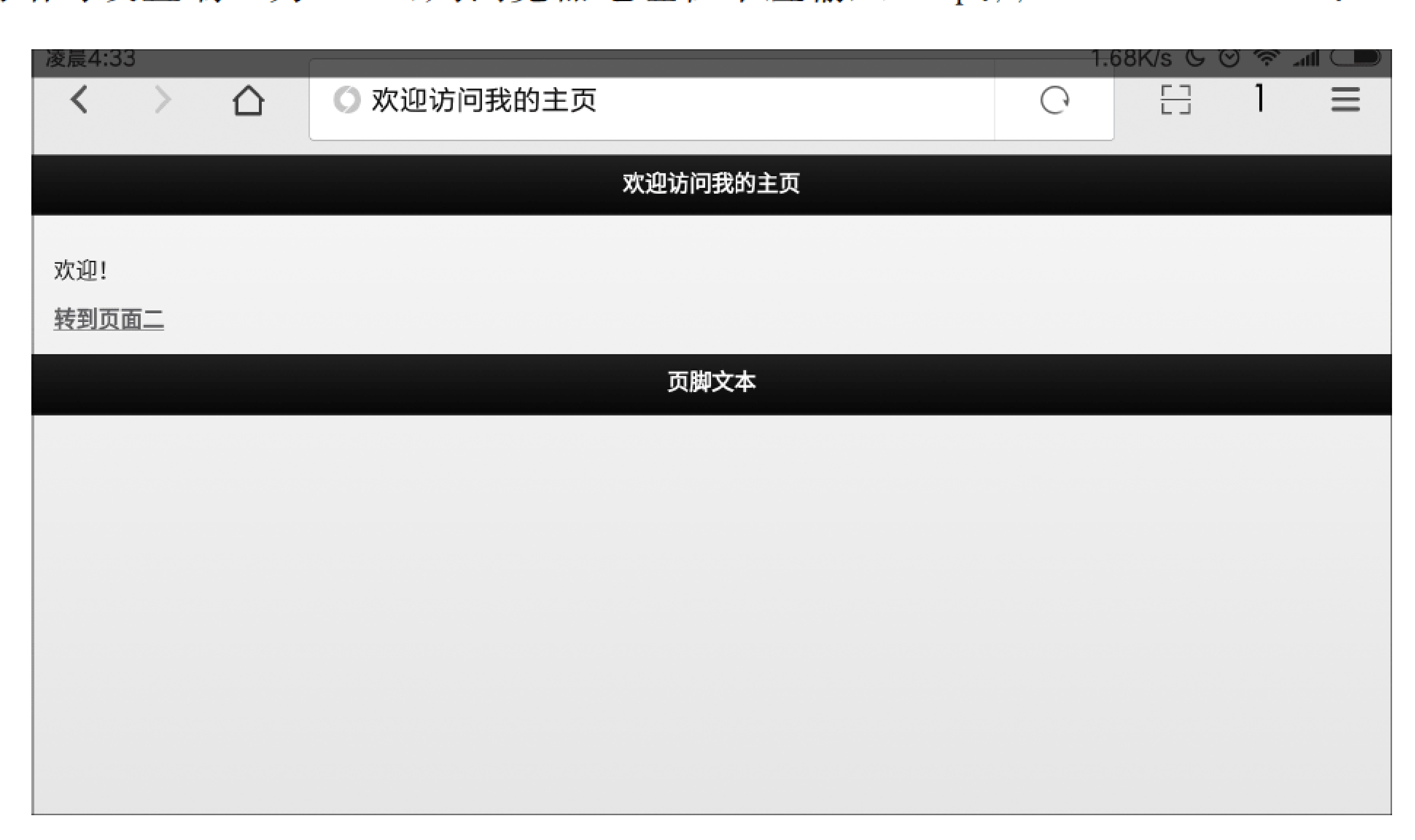


图 9.61 在小米手机浏览器中浏览例 9-1-2-3.htm

9.3 实验任务

1. 实验题目

运用 jQueryMobile 开发手机版画图程序。

2. 程序功能

在实验 6.5 的基础上新增如下功能。

（1）在纵向树形面板新增“速度”选项，内含运动频率和运动步长2个方面，并编程增加圆的静和动2个功能。当在树形面板中选中动，主菜单中选中圆，则圆从左上角到右下角，碰到右下角而反向运动到左上角，碰到左上角后再次反向运动，周而复始直到静按钮被按下。

（2）在左侧树形面板中选择全部用Web存储记录，打开网页后用Web存储初始化当前选项。

3. 实验目的

（1）jQueryMobile布局技术。

（2）掌握jQueryMobile按钮、图标、工具栏、导航栏、列表、事件的运用方法。

（3）掌握运用IIS发布网站的技术。

（4）掌握浏览网页设置Windows防火墙技术。

4. 实验类型

综合设计。

5. 实验要求

（1）脚本程序完全运用jQuery或jQueryMobile程序实现。

（2）运用JSON进行面向对象应用程序开发。

（3）所有画图全部用画布技术实现。

（4）综合运用jQueryMobile按钮、图标、工具栏、导航栏、列表、事件。

6. 实验环境

（1）服务器端

① 计算机：PC、内存8GB、主频1.8GHz及以上、硬盘500GB及以上。

② 操作系统：Windows XP、Windows 7、Windows 8、Windows 10。

③ 开发环境：Visual Studio 2010。

④ Web服务器：IIS。

⑤ 浏览器：Chrome或QQ浏览器。

（2）浏览器端

① 手机、PAD。

② Chrome或Safari。

（3）网络环境

无线上网。

7. 实验原理

给出与事件、列表、图标、导航栏、工具栏、按钮直接相关的jQueryMobile语句，并辅以必要的说明与分析。

8. 遇到的问题及解决办法

（1）给出所遇到的全部错误现象描述。
（2）给出修正错误前设定的断点位置。
（3）给出修正错误所监视的变量。
（4）给出运行到断点处监视变量值。

9. 运行结果

给出所完成任务功能的屏幕截图。

第 10 章 chapter 10

Node.js

■ 知识目标
- 掌握 Node.js 的特性和优点
- 掌握 Node.js 的安装和运行环境设置
- 掌握 Node.js 的 Web 服务器发布以及浏览器与 Node.js 的 Web 服务器间的通信
- 掌握 Node.js 的文件管理技术
- 掌握 Node.js 的多事件处理技术

■ 能力目标
- 能够根据实际情况,将任何浏览器端脚本和网页转成 Node.js 服务器脚本运行
- 能够根据定义恰当的事件并予以处理
- 能够用 Node.js 服务器端文件管理技术进行数据管理

■ 素质目标
- 运用 Node.js 部署高性能的网站

■ 教学重点
- 浏览器与 Node.js 服务器间的通信以及事件定义及处理技术

■ 教学难点
- 部署高性能的网站

■ 建议学时
- 理论：4 学时
- 实验：2 学时

10.1 基础知识

10.1.1 概述

Node.js 是一个 JavaScript 运行环境。它封装了 Google V8 引擎。V8 引擎执行 JavaScript 的速度快、性能好。Node.js 对一些特殊用例进行了优化,提供了替代的 API,

使得 V8 在非浏览器环境下运行得更好。Node.js 基于 Chrome JavaScript 运行时态，便于搭建响应速度快、易于扩展的网络应用。Node.js 基于非阻塞 I/O 模型，使用事件驱动的方式，运行起来轻量和高效，非常适合在分布式设备上运行数据密集型的实时应用。

1. 特性

V8 引擎本身使用了一些最新的编译技术，使得用 JavaScript 这类脚本语言编写出来的代码运行速度获得了极大提升，又节省了开发成本。对性能的苛求是 Node 的一个关键因素。JavaScript 是一个事件驱动语言，Node 利用了这个优点，编写出可扩展性高的服务器。Node 采用了称为事件循环(event loop)的架构，使得编写可扩展性高的服务器变得既容易又安全。提高服务器性能的技巧有多种多样。Node 选择了既能提高性能又能减低开发复杂度的架构。Node 绕过复杂的并发编程，但仍提供很好的性能。

Node 采用一系列“非阻塞”库来支持事件循环的方式，以此为文件系统、数据库之类的资源提供接口。向文件系统发送一个请求时，无须等待硬盘(寻址并检索文件)，硬盘准备好时，非阻塞接口会通知 Node。该模型以可扩展的方式简化了对慢资源的访问。

让 JavaScript 运行于服务器端不是 Node 的独特之处，却是其强大功能之一。浏览器环境限制了选择编程语言的自由。任何服务器与日益复杂的浏览器客户端应用程序间共享代码的愿望只能通过 JavaScript 来实现。虽然还存在其他一些支持 JavaScript 在服务器端运行的平台，但 Node 发展迅猛，已成为广泛使用的平台。

2. 优点

(1) 单线程

Node.js 是单线程的，在不新增额外线程的情况下，可以对任务进行并发处理。它通过事件轮询(event loop)实现并发操作，应用时尽可能避免阻塞操作，使用非阻塞操作。

(2) 模块化

Node.js 使用模块划分不同的功能，以简化应用的开发。模块有类似 C++ 语言中的类库。每一个 Node.js 的类库都包含十分丰富的各类函数，比如 http 模块就包含了和 http 功能相关的很多函数，可以帮助开发者很容易地对比 http、tcp/udp 等操作，还可以很容易地创建 http 和 tcp/udp 的服务器。

在程序中使用模块十分方便。假设引入 http 类库，将对 http 类库的引用存放在 http 变量中。此时，Node.js 会在应用中搜索是否存在 node_modules 的目录，并且搜索这个目录中是否存在 http 的模块。如果 Node.js 找不到这个目录，则会到全局模块缓存中去寻找，用户可以通过相对或者绝对路径指定模块的位置，比如：

```
var myModule=require('./myModule.js');
```

模块中包含了很多功能代码片段，模块中的代码大部分都是私有的。当然，可以将某些方法和变量暴露到模块外，这时可以使用 exports 对象实现。

3. 集成开发环境

具备书写 JavaScript 功能的任何 IDE 或普通的记事本。

4. 应用方向

Node.js 已发展成一个成熟的开发平台，吸引了许多开发者。许多大型高流量网站都采用 Node.js 开发，此外，开发人员还可以使用它来开发一些快速移动的 Web 框架。

除了 Web 应用外，Node.js 也被应用在许多方面，比如监控、媒体流、远程控制、桌面和移动应用等。

10.1.2 Windows 下的安装

Windows 官网提供安装包和编译器，可以在官网下载 32 位或 64 位安装包。

本文在 Windows 8 专业版环境下，以 64 位 v4.4.3 版本为例讲述安装过程，其他版本类似。

1. 下载安装包

打开浏览器(火狐)，在地址栏中输入下载地址，如图 10.1 所示，按下回车键确认，弹出如图 10.2 所示的对话框。在图 10.2 中单击“浏览”按钮，打开图 10.3 所示的对话框。在图 10.3 中选择 F:\教学文档\自编教材\HTML5 实用中网页设计技术\环境文件，单击“选择文件夹”按钮，进入如图 10.4 所示的对话框。在图 10.4 中单击“保存文件”按钮，则成功保存文件，如图 10.5 所示。



图 10.1 在浏览器中输入下载地址



图 10.2 正在打开对话框



图 10.3 “选择下载文件夹”对话框

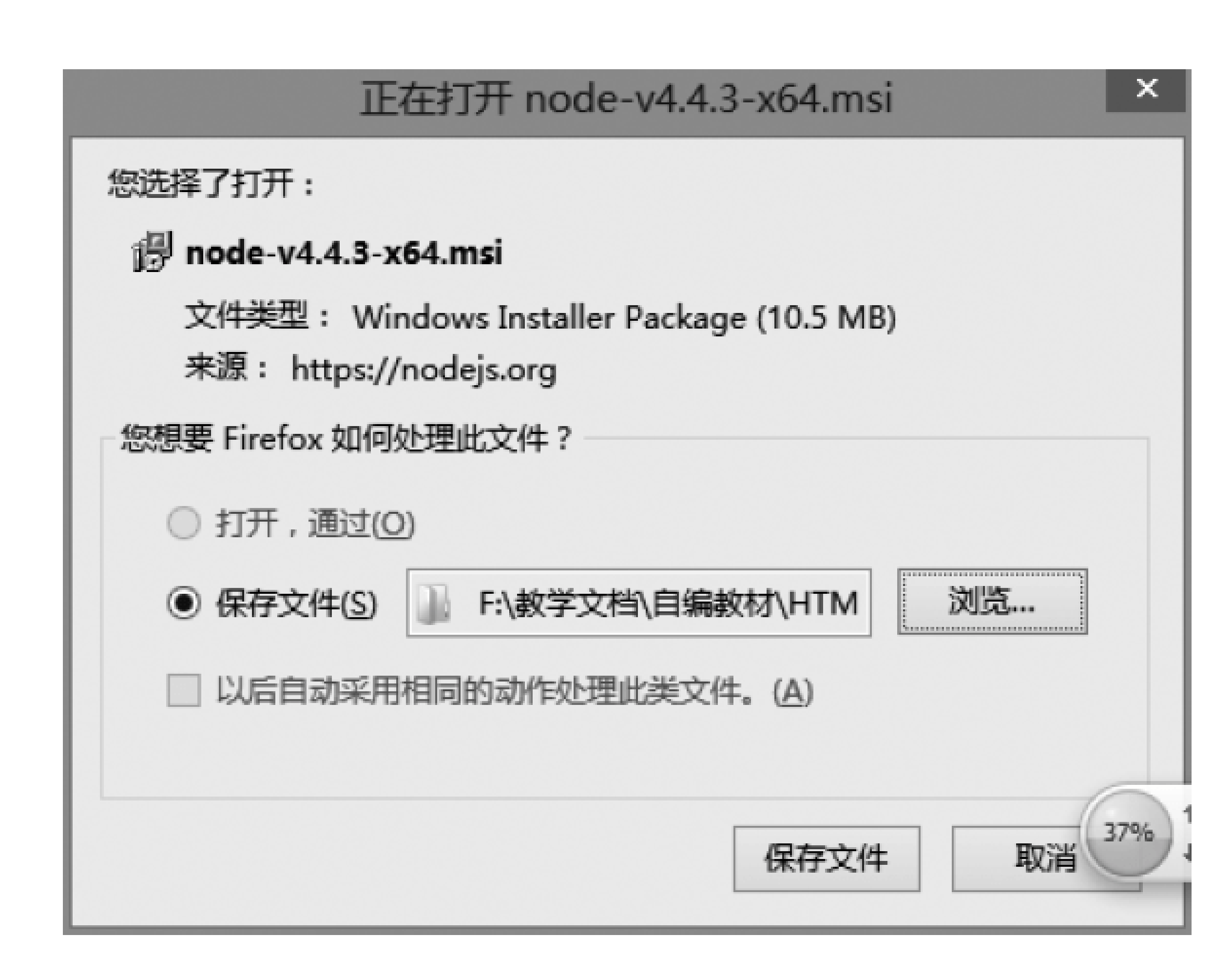


图 10.4 选择文件夹后的正在打开对话框

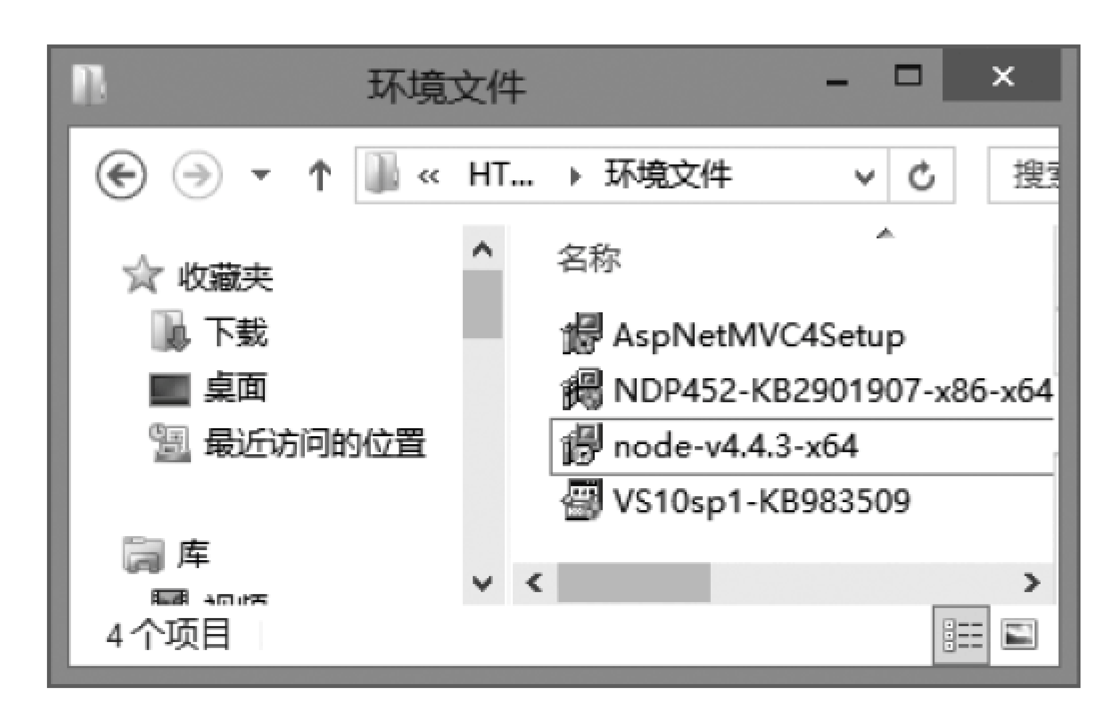


图 10.5　成功下载安装包的环境文件夹

2. 运行安装包

① 单击图 10.5 中的 node-v4.4.3-x64 安装文件，得到图 10.6 所示的安装对话框。

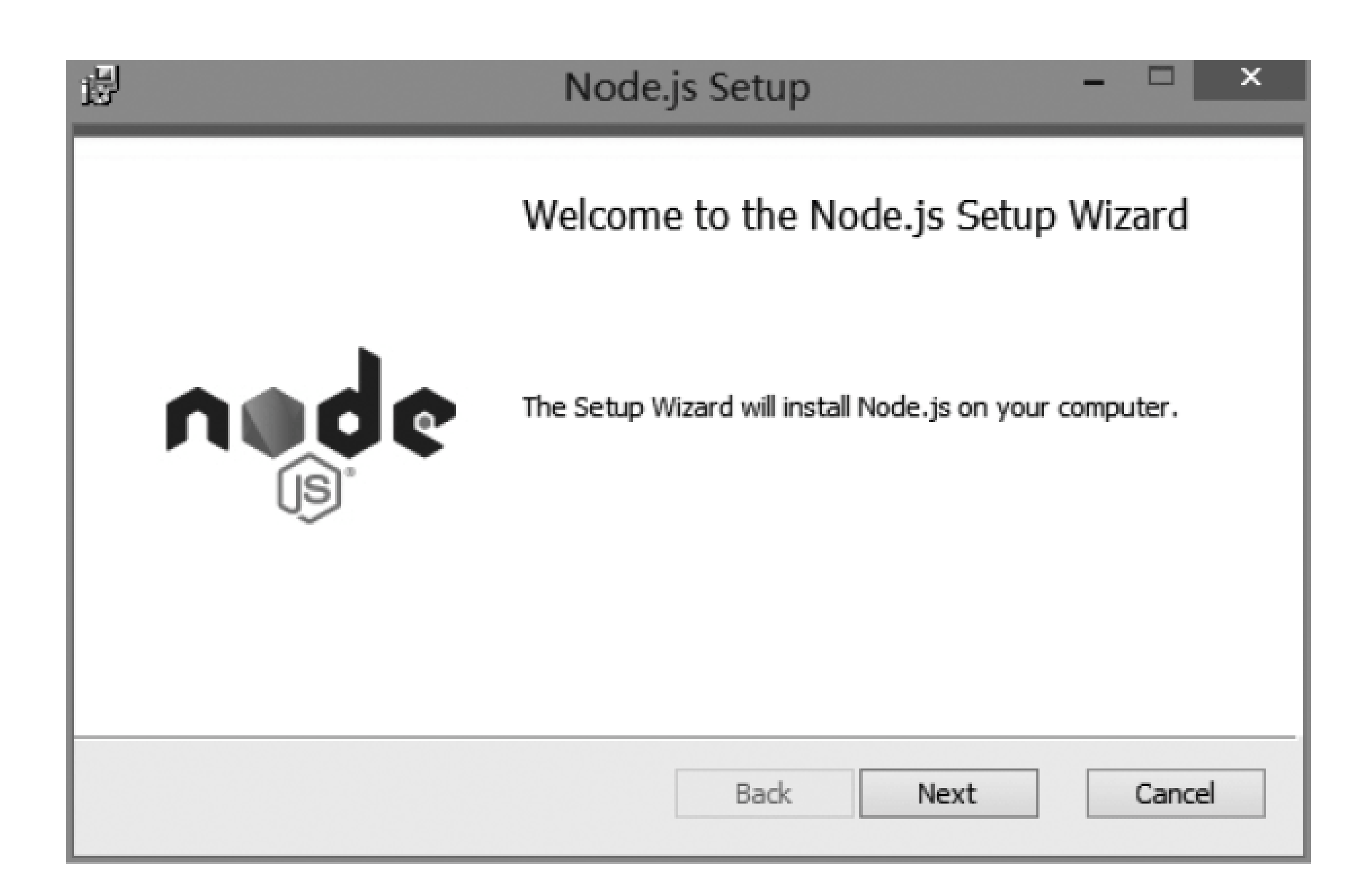


图 10.6　安装对话框

② 单击图 10.6 中的 next 按钮，得到图 10.7 所示的接受许可协议对话框。

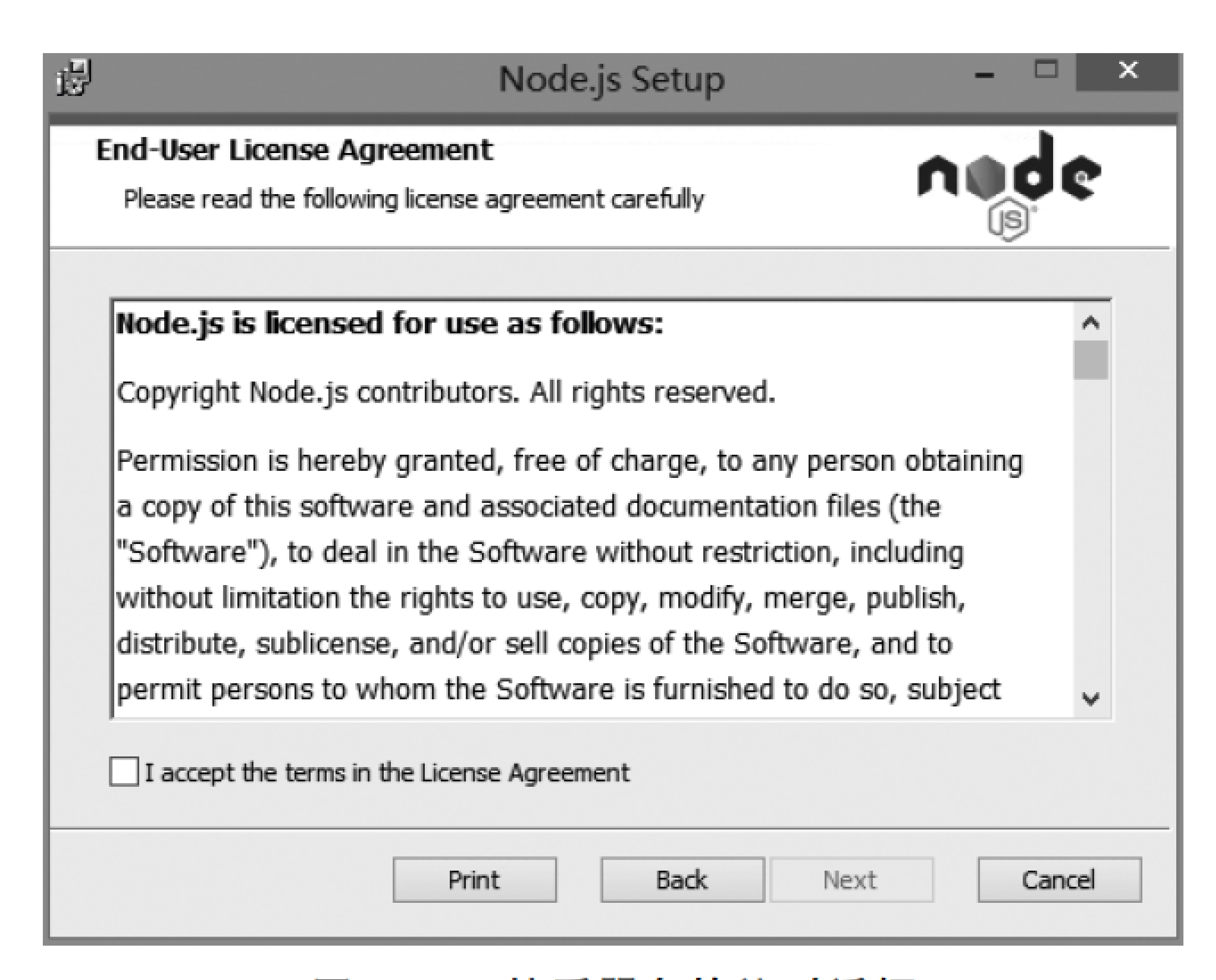


图 10.7　接受服务协议对话框

③ 勾选图 10.7 中的 I accept the terms in the License Agreement，单击 Next 按钮，打开图 10.8 所示的接受许可协议对话框。

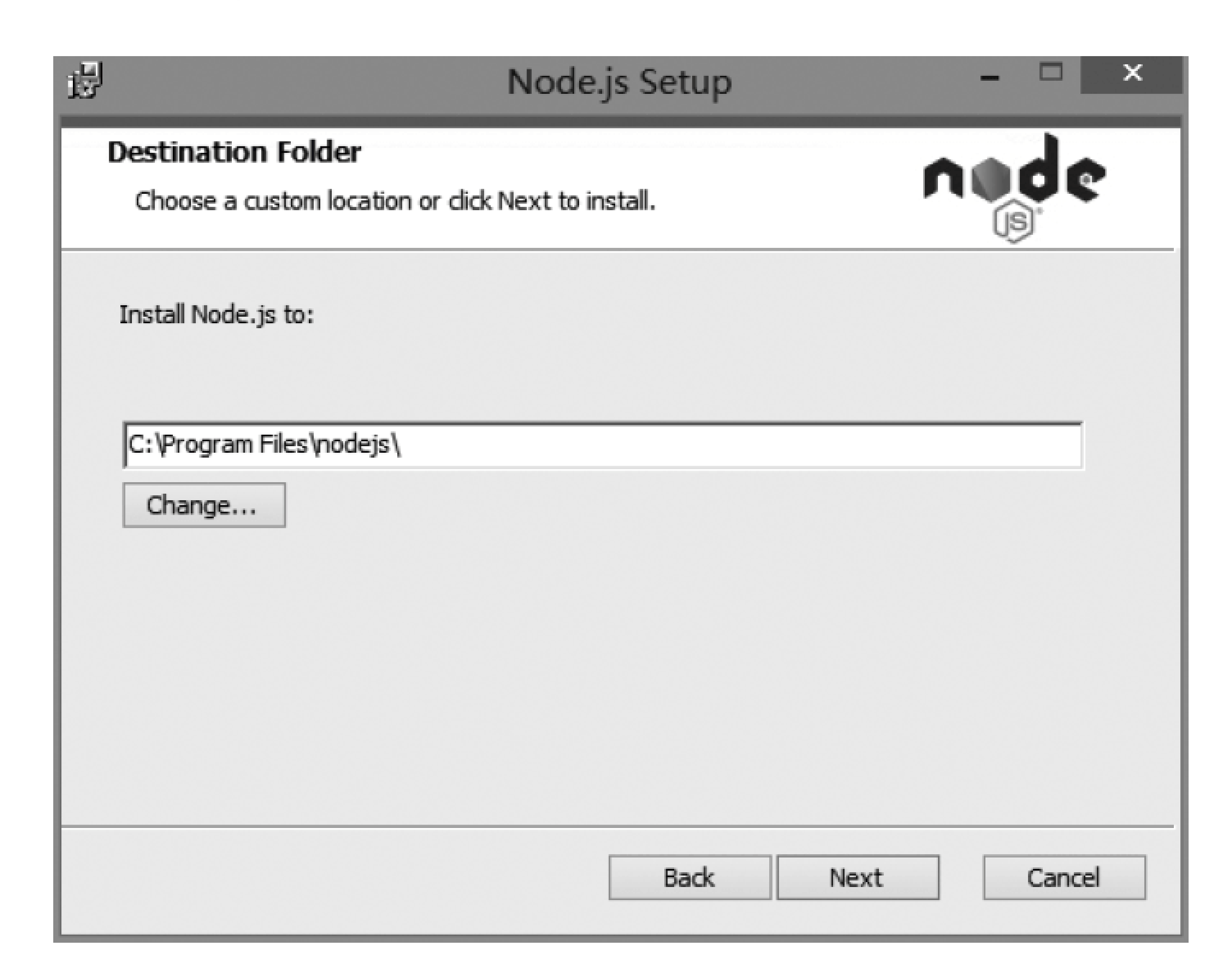


图 10.8　选择安装文件夹对话框

④ 单击图 10.8 中的 Next 按钮，打开图 10.9 所示的选择功能模块对话框。

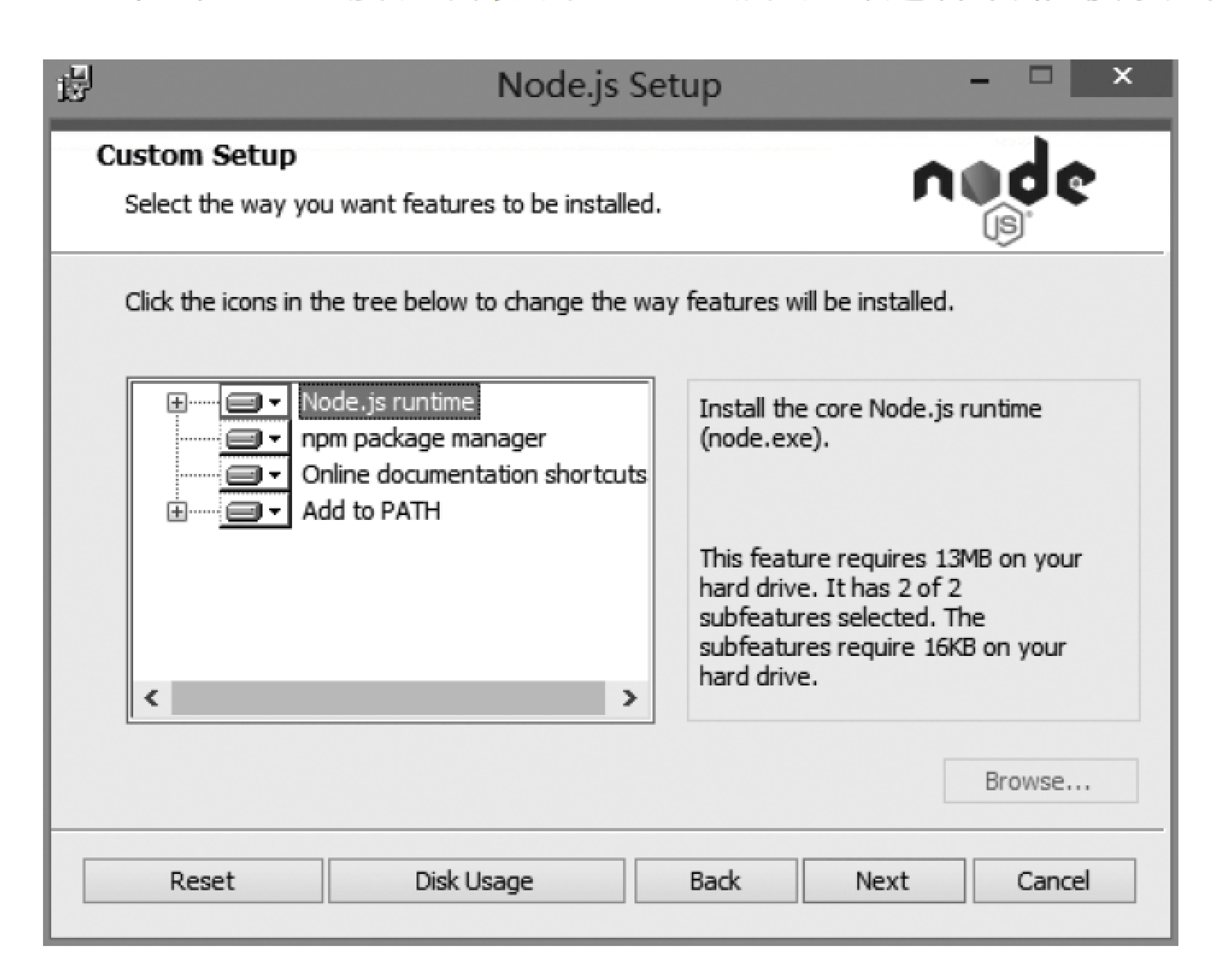


图 10.9　选择功能模块对话框

⑤ 单击图 10.9 中的 Next 按钮，打开图 10.10 所示的准备安装对话框。

⑥ 单击图 10.10 中的 Next 按钮，打开图 10.11 所示的正在安装初态对话框。随着安装不断进展，会看到图 10.12 所示的对话框。安装完成，弹出图 10.13 所示的安装结束对话框。

⑦ 单击图 10.13 所示的 Finish 按钮，确认安装完成。

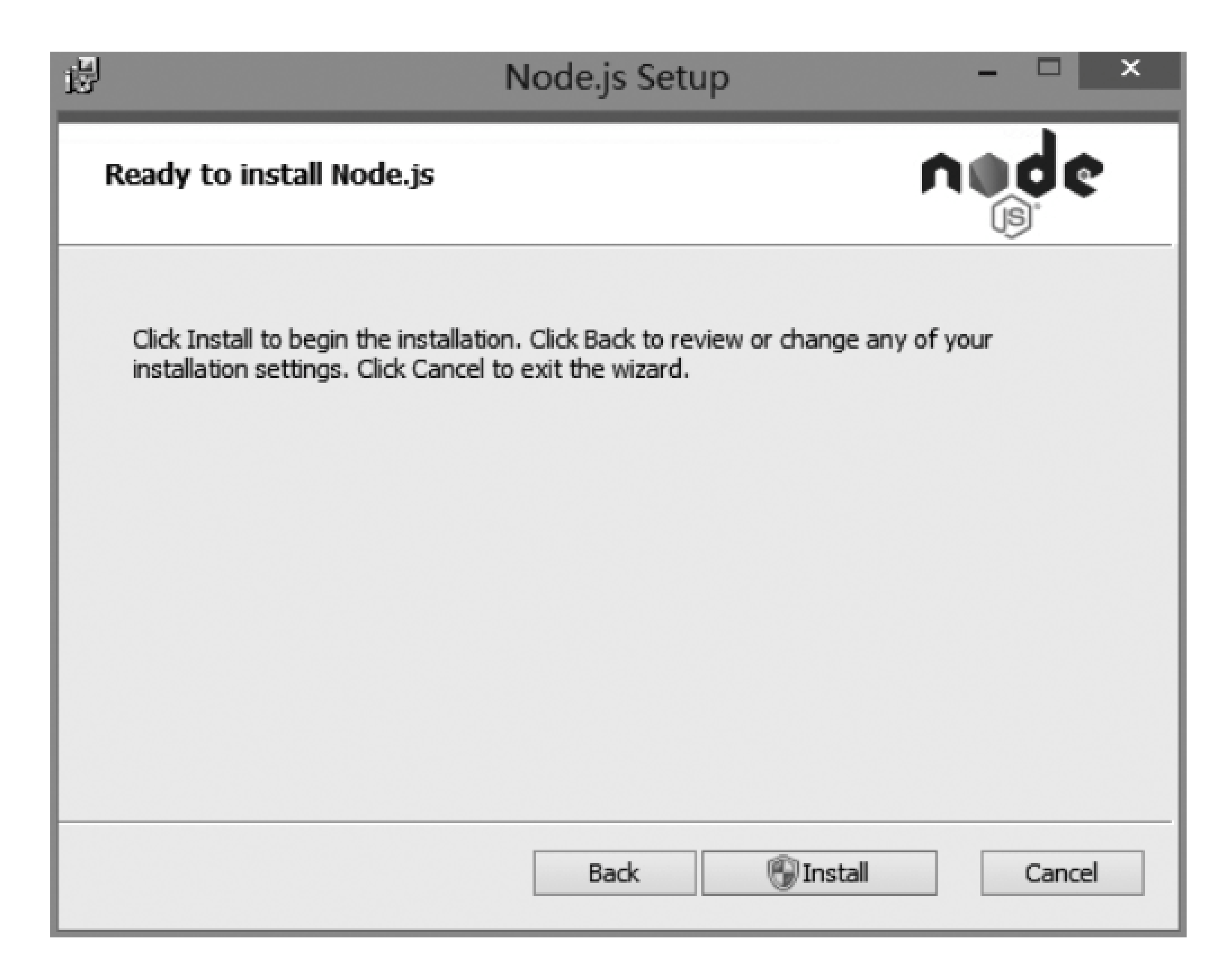


图 10.10　准备安装对话框

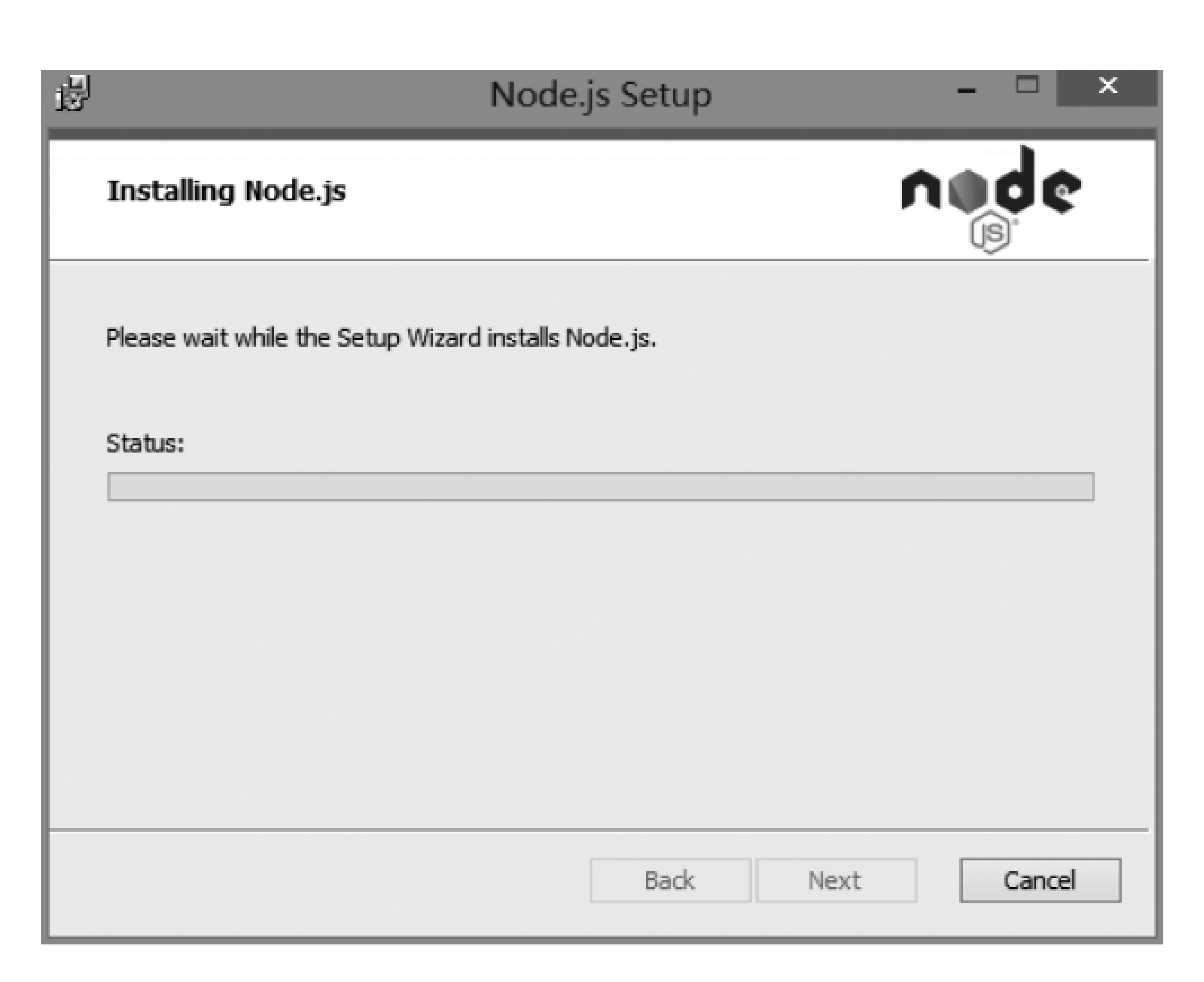


图 10.11　正在安装初态对话框

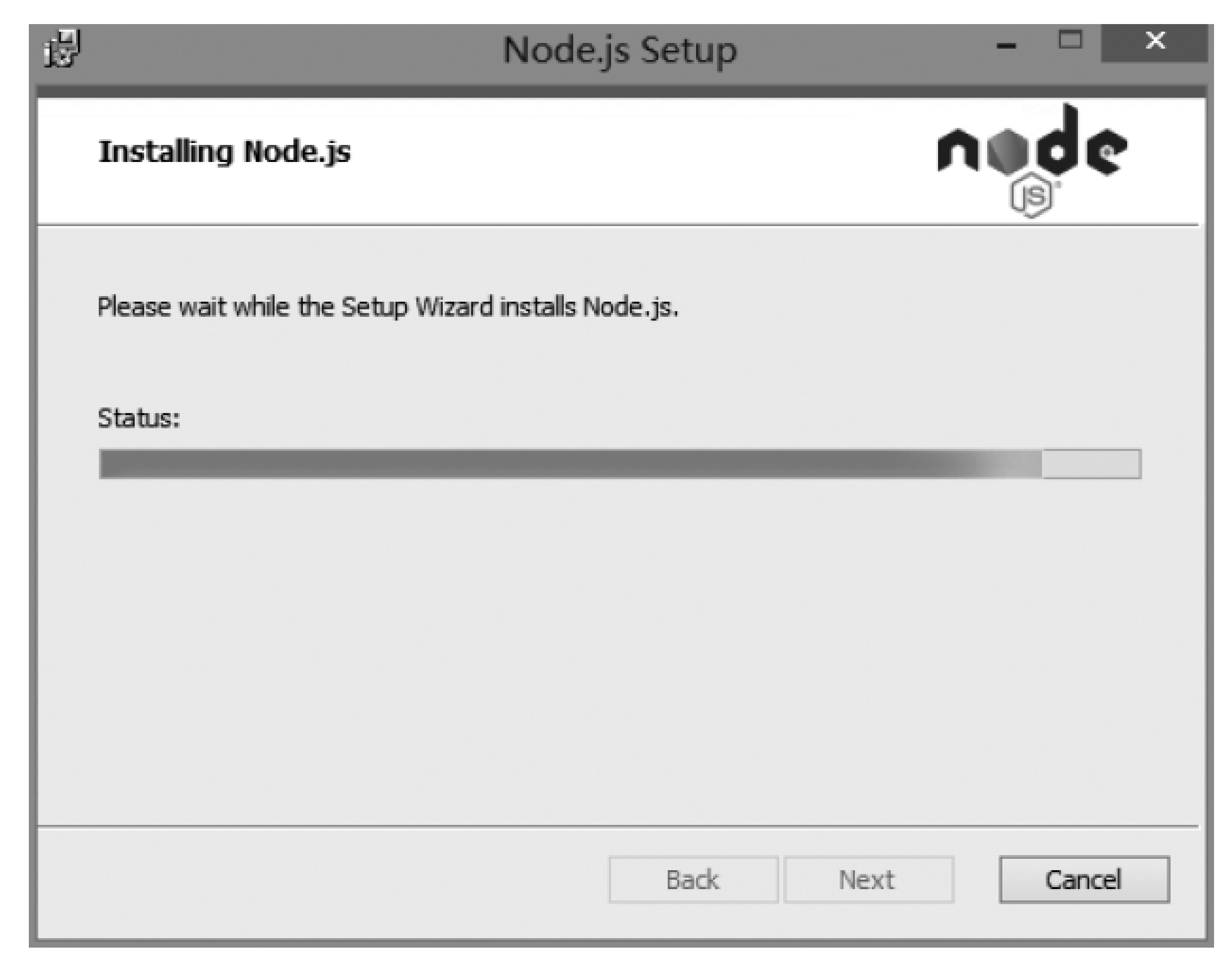


图 10.12　安装接近结束的对话框



图 10.13 安装结束对话框

3. 确认安装成功

单击 Windows 8 的“开始”菜单，展开“所有程序”，拖动滚动条，找到“Node.js 文件夹”，展开 Node.js 文件夹，得到如图 10.14 所示的程序界面，表明安装成功。

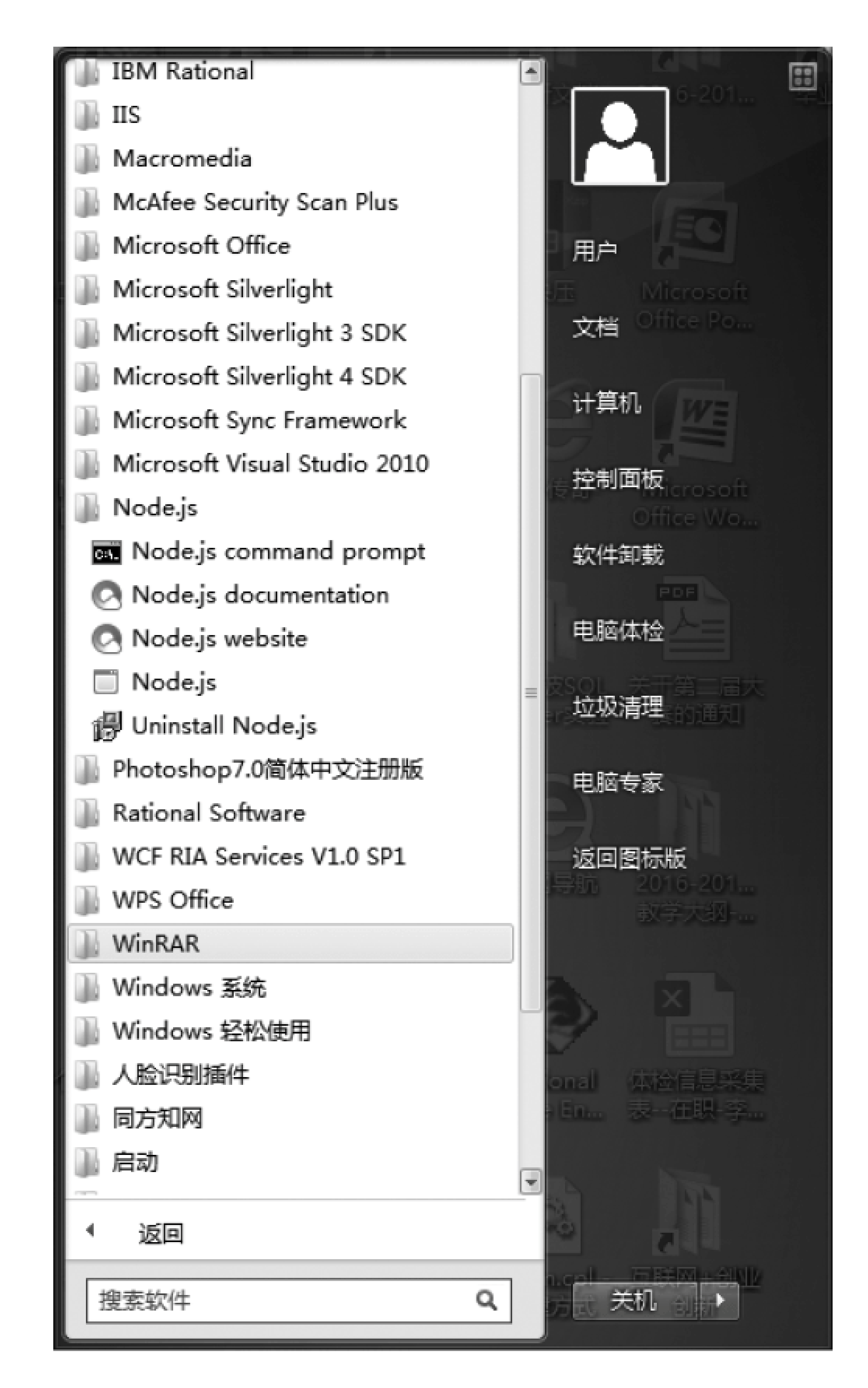


图 10.14 成功安装的“开始”菜单程序组

4. 核准安装版本

单击 Windows 8 的“开始”菜单，得到如图 10.15 所示的对话框。在搜索软件栏目中输入“运行”，筛出“运行”程序，如图 10.16 所示。单击图 10.16 所示的“运行”程序，进入图 10.17 所示的界面。在图 10.17 中输入 cmd，单击“确定”按钮，得到图 10.18 所示的 DOS 命令提示符对话框。在图 10.18 中进入安装文件所在目录，输入 node --version 命令，在控制台命令行输出 v4.4.3，如图 10.19 所示。



图 10.15 “开始”菜单



图 10.16 筛选出“运行”程序的程序组

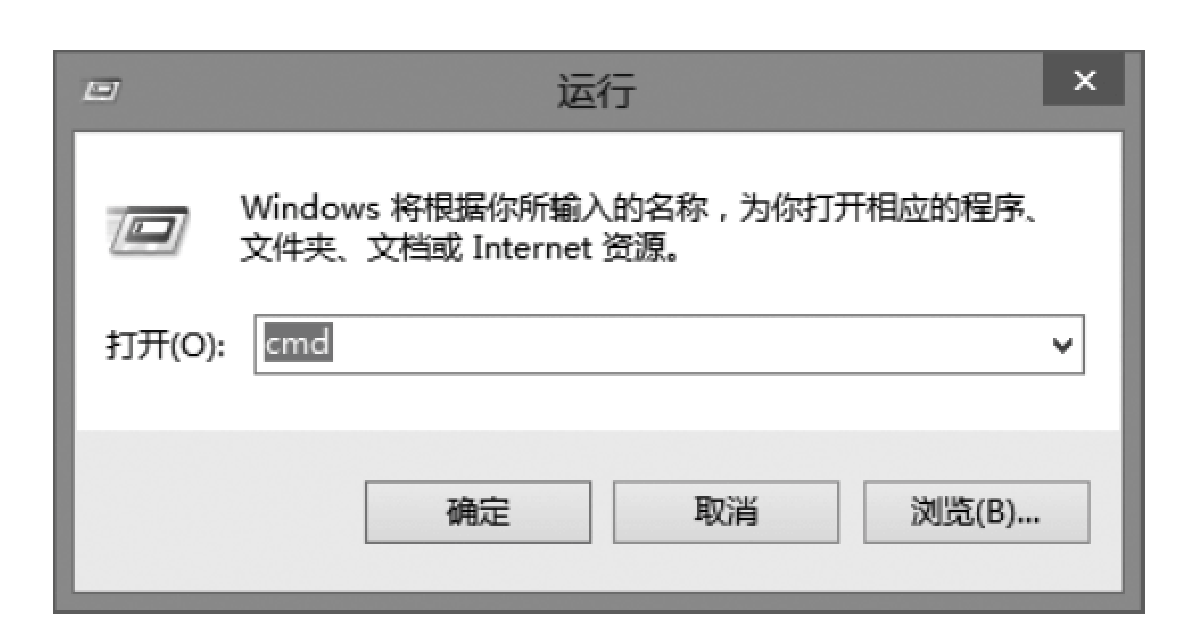


图 10.17 运行程序界面

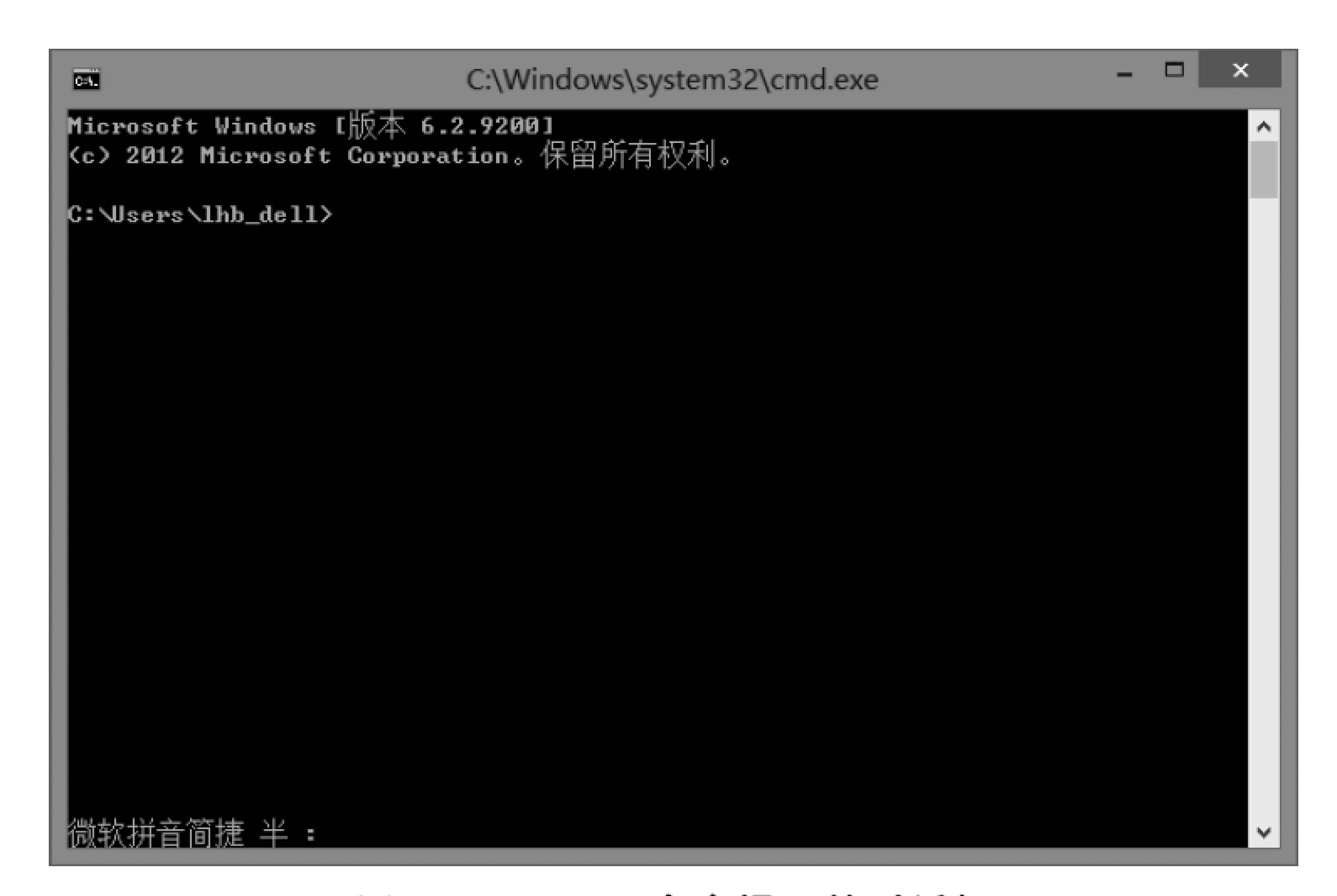


图 10.18 DOS 命令提示符对话框

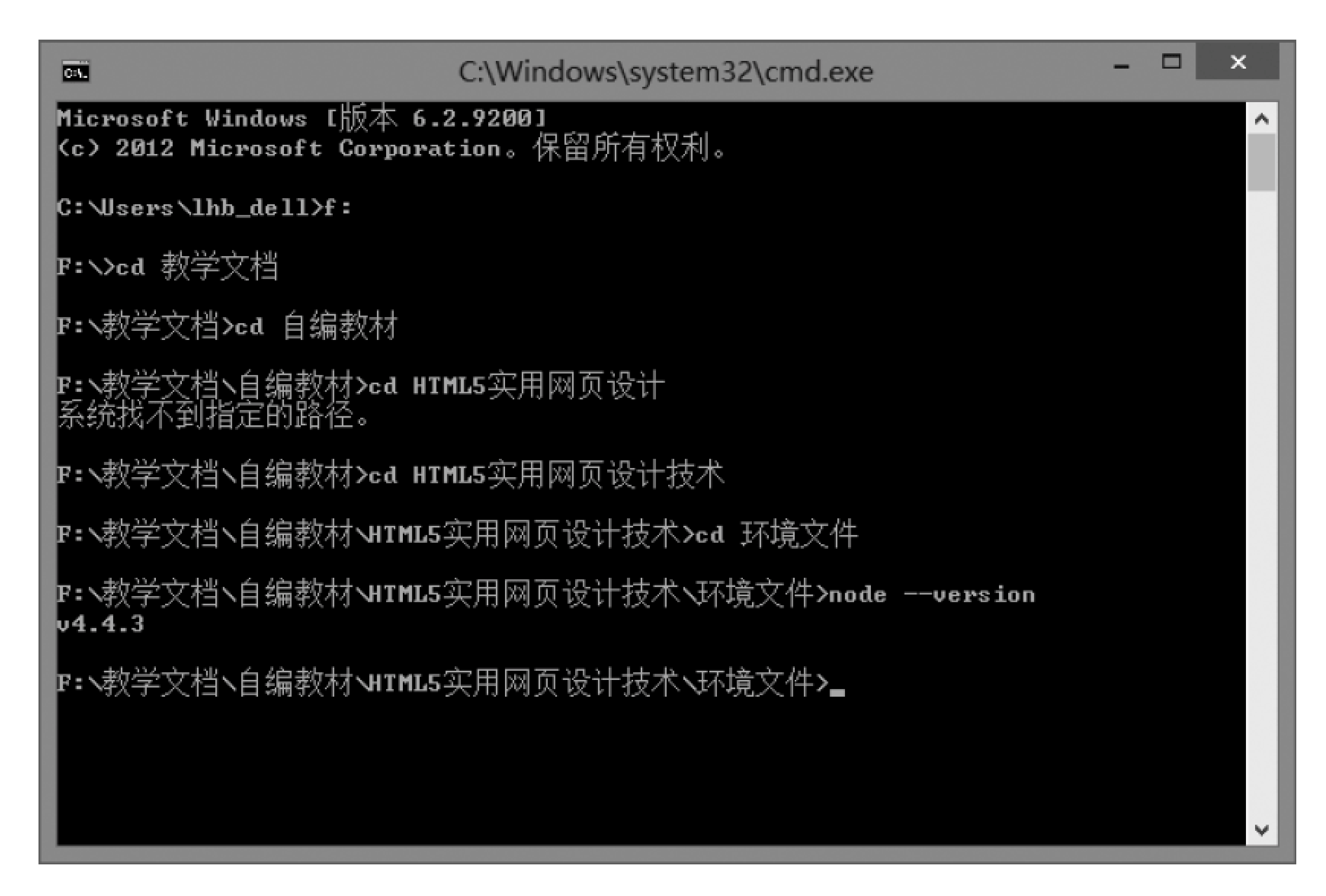


图 10.19 查实的 Node.js 版本对话框

10.1.3 一个简单的 Hello World 输出示例

在 Node 中，Web 应用是首要的。Node 优化了服务器的 http 创建。下面以一个运用 Node.js 的在网页中显示 Hello World 的样例程序为例说明应用的步骤。

1. 网络环境

运行 Node.js 服务器和浏览网页的客户机属于同一台机器，IP 地址为 192.168.31.176。

2. 启动 Node.js 的 http 服务

(1) 建立脚本文件

打开 VS 2010 的源程序网站解决方案，将例 10-1-3.js 文件建立在图 10.20 所示的路径下。



图 10.20 例 10-1-3.js 文件所在的路径

(2) 例 10-1-3.js 源文件内容

```
var http=require('http');
server=http.createServer(function (req, res) {
    res.writeHeader(200, {"Content-Type": "text/plain"});
    res.end("Hello World\n");
});
server.listen(8000);
console.log("httpd start @8000");
```

(3) 在 Node.js 服务中启动例 10-1-3.js 的 http 服务

① 启动 Node.js command prompt。

② 单击“开始”菜单，展开所有程序，展开 Node.js，如图 10.21 所示。选择 Node.js command prompt，输入命令 node F:\教学文档\自编教材\HTML5 实用网页设计技术\源程序\理论\第 10 章-Node\例 10-1-3.js，按下回车键，执行命令的结果如图 10.22 所示。在图 10.22 所示的控制台中输出执行提示 httpd start @8000，说明 http 服务已经启动，等待浏览器访问。

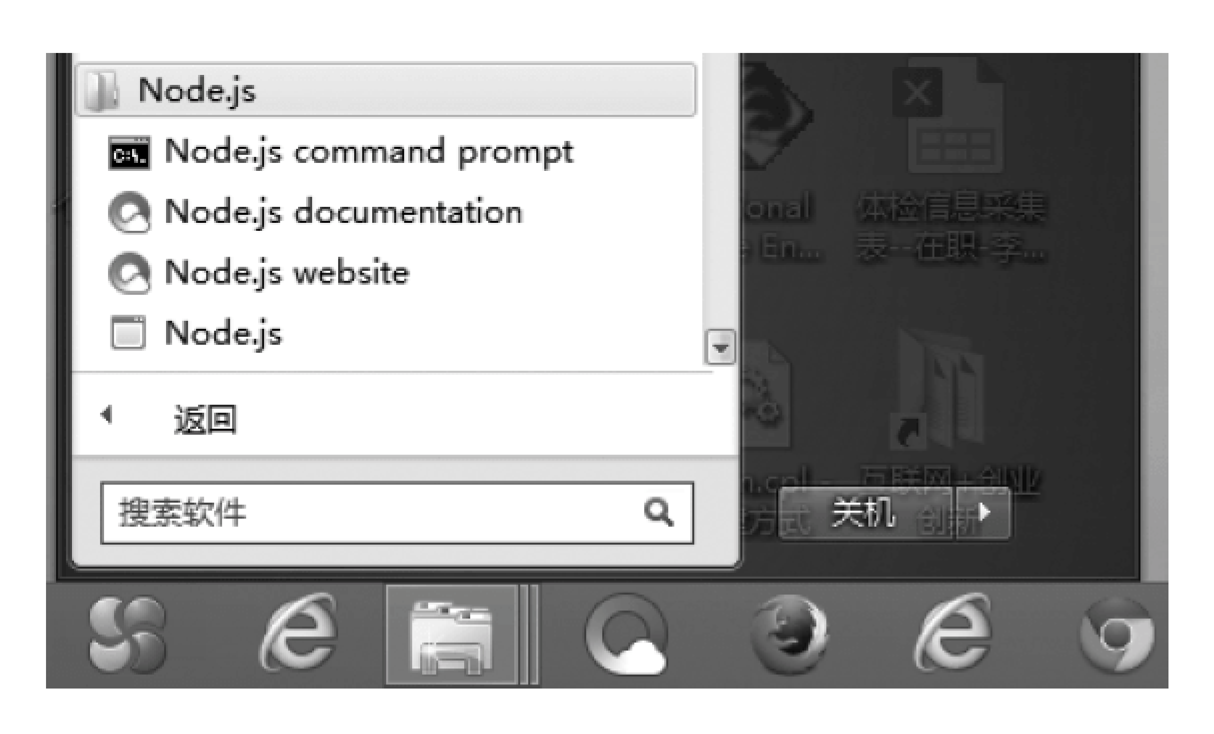


图 10.21　展开 Node.js 的开始菜单

```
node  F:\教学文档\自编教材\HTML5实用网页设计技术\源程序\理论\第10章-N...
Your environment has been set up for using Node.js 4.4.3 (x64) and npm.

C:\Users\lhb_dell>node F:\教学文档\自编教材\HTML5实用网页设计技术\源程序\理论\第
10章-Node\例10-1-3.js
httpd start @8000
```

图 10.22　展开 Node.js 的开始菜单

3. 浏览器访问：192.168.31.176:8000

打开谷歌浏览器，在地址栏中输入 192.168.31.176:8000，按回车键前往该地址端口，在网页中输出 Hello World，如图 10.23 所示。



图 10.23　在谷歌浏览器中访问 192.168.31.176:8000 的显示结果

4. 程序解析

Node 网络应用需要先通过 createServer 创建一个网络服务对象。

传入 createServer 的 function 在每次 http 请求时都将被调用执行，因此这个 function 也被称为请求的处理者。当 http 请求这个服务时，Node 调用请求处理者

function 并传入一些用于处理事务相关的对象：request 和 response。实际上，通过 createServer 返回的 Server 对象是一个 EventEmitter，用户应用程序需要保存 Server 对象，并在之后对其添加监听器。

为了监听到服务请求，在 Server 对象上需要调用 listen 方法。绝大多数情况需要传给 listen 服务监听的端口号。

10.2 Web 应用基础

10.2.1 Web 服务器

1. 定义

Web 服务器一般指网站服务器，是指驻留于因特网上某种类型计算机的程序。Web 服务器的基本功能是提供 Web 信息浏览服务，它支持 http 协议、HTML 文档格式及 URL，能与客户端的网络浏览器配合。

大多数 Web 服务器都支持服务端的脚本语言，并通过脚本语言从数据库获取数据，将结果返回给客户端浏览器。

目前最主流的 3 个 Web 服务器是 Apache、Nginx、IIS。

2. 架构

Web 应用的典型架构如图 10.24 所示。

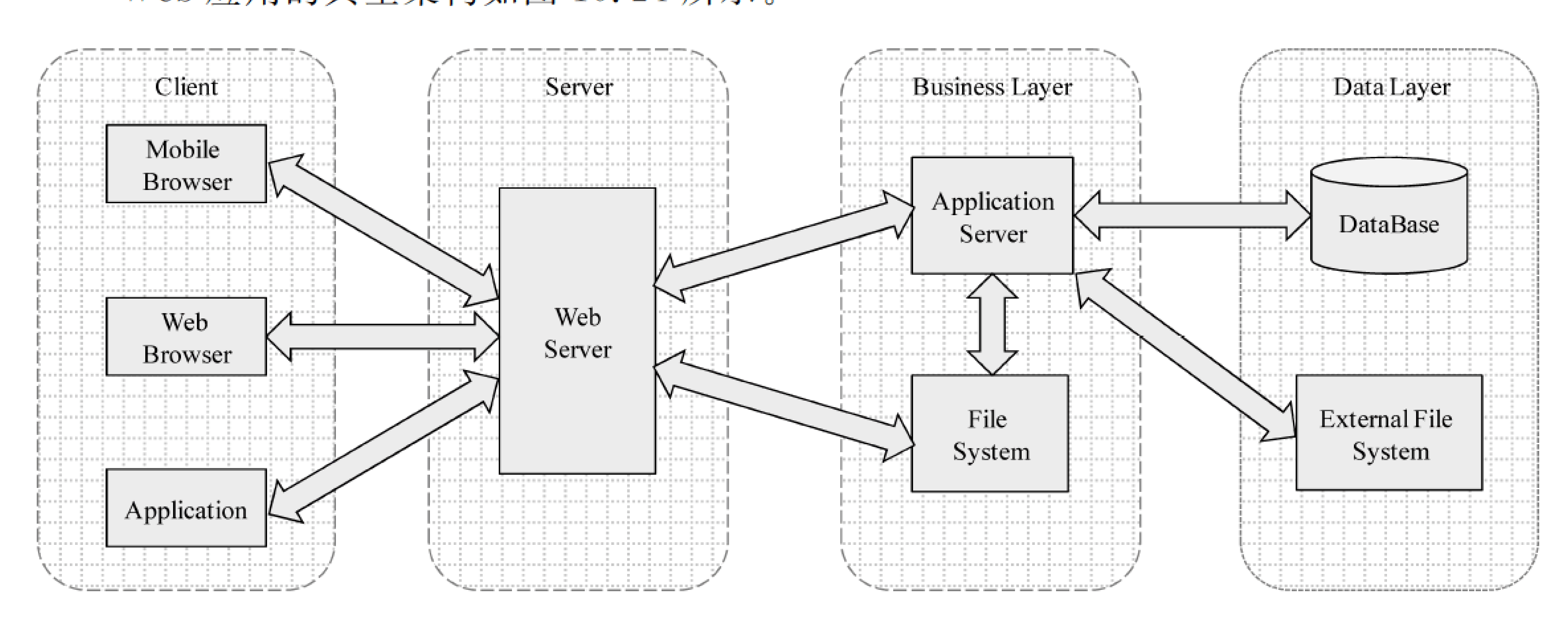


图 10.24 Web 应用的典型架构

Client：客户端，一般指 Web 浏览器，可以通过 HTTP 协议向服务器请求数据。

Server：Web 服务器，可以接收 Web 浏览器的请求，并向浏览器发送响应数据。

Business Layer：业务层，通过 Web 服务器处理应用程序，如与数据库交互、逻辑运算、调用外部程序等。

Data Layer：数据层，一般由数据库组成。

3. 创建

Node.js 提供了 http 模块，主要用于搭建 HTTP 服务端和客户端，使用 HTTP 服务器或客户端功能必须调用 http 模块，代码为

```
var http=require('http');
```

例 10-2-1-1 一个最基本的 Web 服务器架构应用示例。

(1) 部署 Web 服务器

① 服务器文件及路径。

Web 服务器路径及相关文件如图 10.25 所示，与本例相关的有例 10-2-1-1.js 文件和 Index.htm 文件。

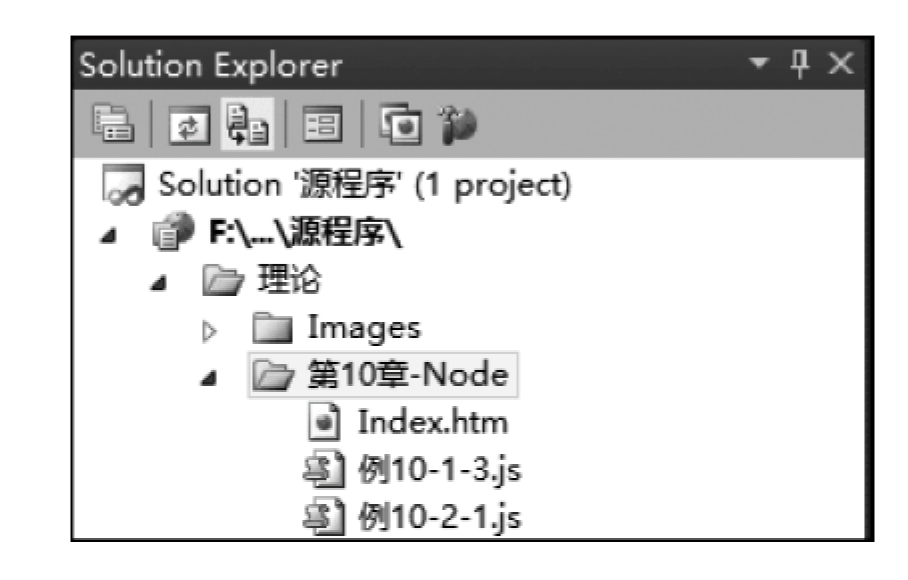


图 10.25 Web 服务器路径及相关文件

② 服务器文件内容。

例 10-2-1-1.js 文件的文件内容如下。

```
var http=require('http');
var fs=require('fs');
var url=require('url');
//创建服务器
http.createServer(function (request, response) {
    //解析请求,包括文件名
    var pathname=url.parse(request.url).pathname;
    //输出请求的文件名
    console.log("Request for "+pathname+" received.");
    //从文件系统中读取请求的文件内容
    fs.readFile(pathname.substr(1), function (err, data) {
        if (err) {
            console.log(err);
            //HTTP 状态码: 404 : NOT FOUND
            //Content Type: text/plain
            response.writeHead(404, { 'Content-Type': 'text/html' });
        }
        else {
            //HTTP 状态码: 200 : OK
            //Content Type: text/plain
            response.writeHead(200, { 'Content-Type': 'text/html' });
            //响应文件内容
            response.write(data.toString());
        }
        //发送响应数据
        response.end();
    });
}).listen(8081);
```

```
//控制台会输出以下信息
console.log('Server running at http://192.168.31.176:8081/');
Index.htm文件内容如下：
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>NodeJS部署Web服务器演示</title>
    </head>
    <body>
        Hello World!
    </body>
</html>
```

③ 启动Web服务器。

打开Node.js命令提示符窗口，在窗口提示符下分别输入如下3条命令。

```
cd F:\教学文档\自编教材\HTML5实用网页设计技术\源程序\理论\第10章-Node
f:
node 例10-2-1-1.js
```

输入每条命令后按回车键执行，得到图10.26所示的成功启动Web服务器状态。



图10.26　成功启动Web服务器的命令提示符窗口

(2) 在浏览器中访问Index.htm

打开火狐浏览器，在地址栏中输入http://192.168.31.176:8081/index.htm，按回车键加载该网页，浏览器显示如图10.27所示。而服务器端的Node.js命令行提示窗口输出信息如图10.28所示。



图10.27　在浏览器中访问Web服务器的Index.htm显示结果



图 10.28 在浏览器中访问 Web 服务器的 Index.htm 服务器端命令提示符窗口显示结果

10.2.2 GET/POST 请求

在很多场景中，Web 服务器需要跟用户的浏览器打交道，如表单提交。表单提交到服务器，一般都使用 GET/POST 请求。下面介绍 Node.js 的 GET/POST 请求。

1. 解析并响应 GET 请求

由于 GET 请求直接被嵌入在路径中，URL 是完整的请求路径，包括了“?”后面的部分，因此程序可以手动解析后面的内容，作为 GET 请求的参数。Node.js 中 URL 模块中的 parse 函数提供了这个功能。

值得注意的是，想要 Node.js 正常显示中文，需要以下两点。

① 将 js 文件保存为 Unicode 格式。判别 js 文件为 Unicode 格式的一个简单方法是使用记事本来判断。使用记事本打开 JS 文件，单击菜单中的“另存为”，看编码格式是否为 UTF-8。若不是，可使用 UltraEdit 工具进行转换，使用记事本也可以转换。

② 在 js 文件中增加编码说明 Meta 数据，让浏览器知道使用什么编码来解释网页。

例 10-2-2-1 一个简单 GET 请求解析及处理中文的示例。

（1）服务器端源程序文件：例 10-2-2-1.js 文件

```
var http=require('http');
var url=require('url');
var util=require('util');
http.createServer(function (req, res) {
    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.write('<head><meta charset="utf-8"/></head>');
    //解析 url 参数
    var params=url.parse(req.url, true).query;
    res.write("你的名字："+params.name);
    res.write("\n");
    res.write("你的年龄："+params.age);
    res.end();
}).listen(3000);
```

(2) 发布 Web 应用

发布 Web 应用服务器的命令如图 10.29 所示。

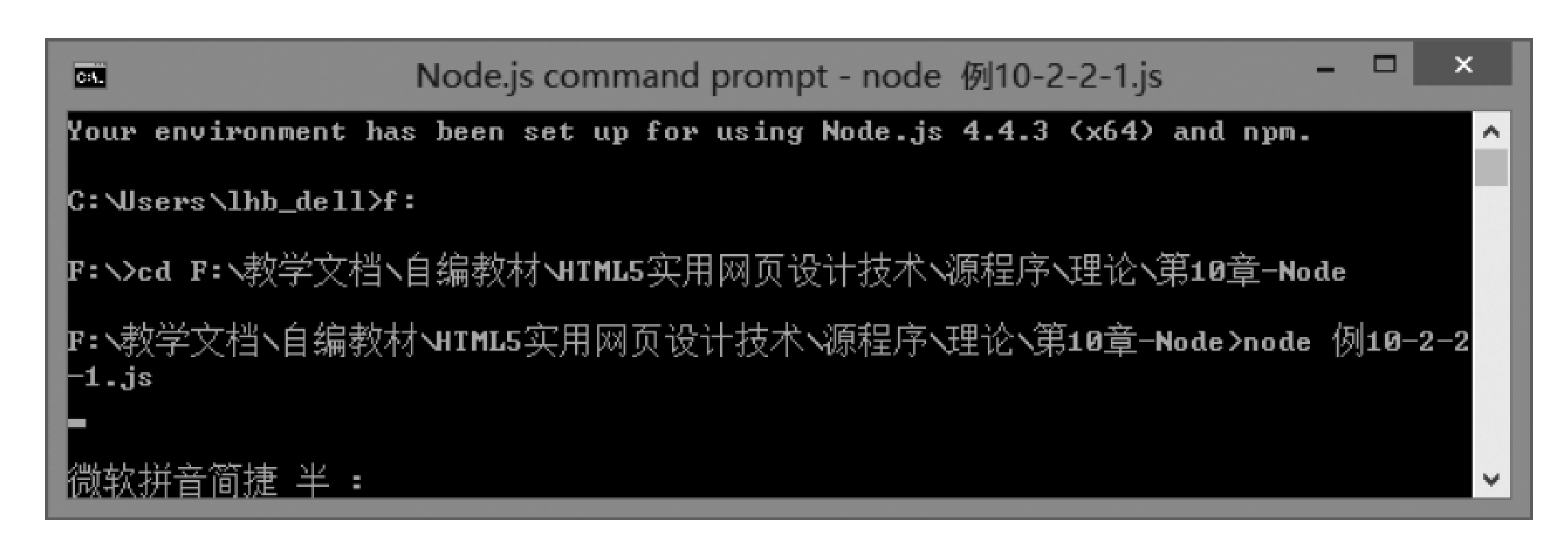


图 10.29　发布 Web 应用 Node.js 的命令提示符 DOS 命令

(3) 浏览器通过 GET 请求访问

浏览器以 GET 方式发起请求及处理结果如图 10.30 所示。



图 10.30　浏览器前往地址栏中的 URL 执行结果

2. 解析并响应 POST 请求

POST 请求的内容全部都在对应的 Form 表单中，http.ServerRequest 并没有一个属性内容为请求体，原因是等待请求体传输可能是一件耗时的工作。

比如上传文件，很多时候程序可能并不需要理会请求体的内容，恶意的 POST 请求会大大消耗服务器的资源，所有 Node.js 默认不会解析请求体，需要时，编程实现解析请求体。POST 请求解析及处理的基本语法结构如下。

```
var http=require('http');
var querystring=require('querystring');
http.createServer(function(req, res){
    //定义了一个 post 变量，用于暂存请求体的信息
    var post='';
    //通过 req 的 data 事件监听函数，每当接受到请求体的数据，就累加到 post 变量中
    req.on('data', function(chunk){
        post+=chunk;
    });
    //在 end 事件触发后，通过 querystring.parse 解析 POST 请求，然后向客户端返回。
    req.on('end', function(){
        post=querystring.parse(post);
```

```
            res.end(util.inspect(post));
        });
}).listen(3000);
```

例 10-2-2-2 一个简单 POST 请求解析及处理中文的示例。

（1）服务器端源程序文件：例 10-2-2-2.js 文件

```
var http=require('http');
var querystring=require('querystring');
var postHTML=
  '<html><head><meta charset="utf-8"><title>服务器解析与处理 POST 请求示例
  </title></head>'+
  '<body>'+
  '<form method="post">'+
  '姓名：  <input name="name"><br>'+
  '年龄：  <input name="age"><br>'+
  '<input type="submit">'+
  '</form>'+
  '</body></html>';
http.createServer(function (req, res) {
    var body="";
    req.on('data', function (chunk) {
        body+=chunk;
    });
    req.on('end', function () {
        //解析参数
        body=querystring.parse(body);
        //设置响应头部信息及编码
        res.writeHead(200, { 'Content-Type': 'text/html; charset=utf8' });

        if (body.name && body.age) { //输出提交的数据
            res.write("姓名："+body.name);
            res.write("<br>");
            res.write("年龄："+body.age);
        } else {  //输出表单
            res.write(postHTML);
        }
        res.end();
    });
}).listen(8000);
```

（2）发布 Web 应用

发布 Web 应用服务器的命令如图 10.31 所示。

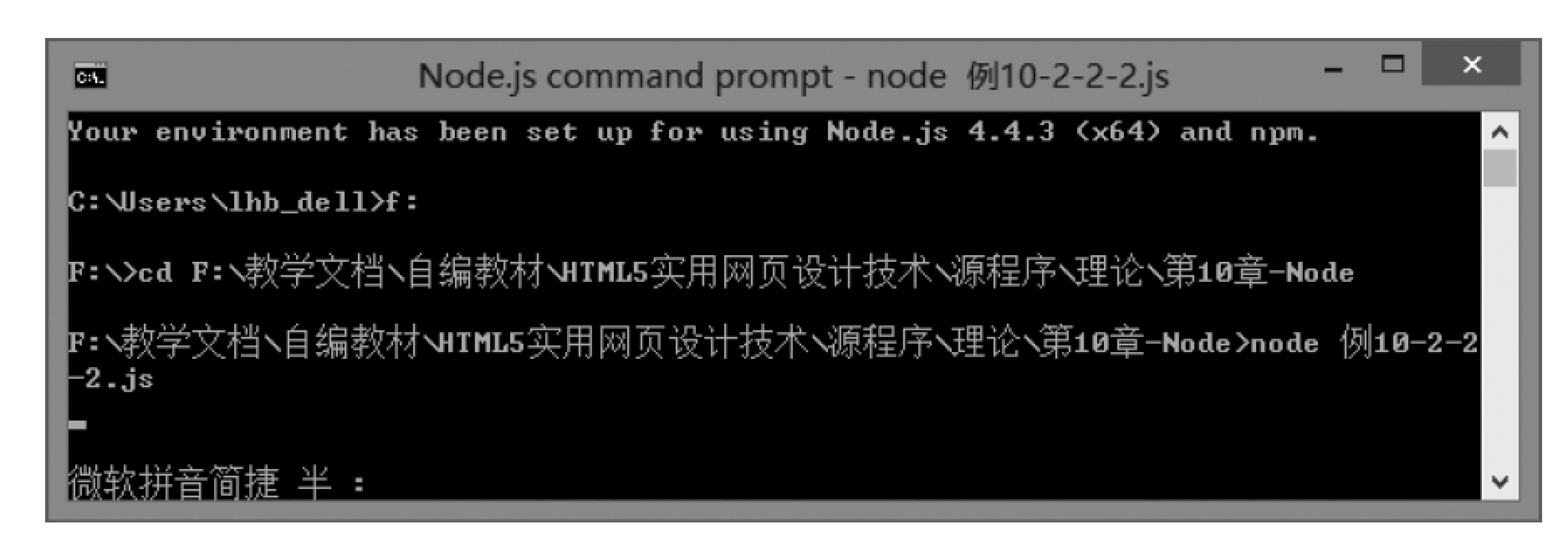


图 10.31 发布 Web 应用 Node.js 的命令提示符 DOS 命令

(3) 浏览器通过 POST 请求访问

用户浏览器向 192.168.31.176:8000 发起请求的执行结果如图 10.32 所示。在图 10.32中输入图 10.33 所示的信息,单击"提交"按钮,服务器响应页面如图 10.34 所示。



图 10.32 浏览器前往地址栏中的 URL 执行结果



图 10.33 输入姓名和年龄的网页状态



图 10.34 单击"提交"按钮后服务器的响应页面

10.3 文件系统

Node.js 提供一组类似 UNIX(POSIX)标准的文件操作 API。Node 导入文件系统模块(fs)的语法如下所示。

```
var fs=require("fs")
```

10.3.1 同步和异步

Node.js 文件系统模块中方法的所有版本均支持异步和同步,例如读取文件内容的函数有异步的 fs.readFile()和同步的 fs.readFileSync()。

1. 异步

异步方法比同步方法性能更高,速度更快,而且没有阻塞。异步的方法函数最后一个参数为回调函数,回调函数的第一个参数包含了错误信息(error)。

例 10-3-1-1 一个简单异步的读取文件示例

(1) 创建例 10-3-1-1.txt 文件

文件内容:明月几时有,把酒问苍天。

(2) 创建例 10-3-1-1.js 文件

源程序如下。

```
var fs=require("fs");
//异步读取
console.log("启动读取文件");
fs.readFile('例 10-3-1-1.txt', function (err, data) {
    if (err)  return console.error(err);
    console.log("------------------------------------");
    console.log("异步读取: "+data.toString());
    console.log("------------------------------------");
});
console.log("程序执行完毕");
```

(3) 执行结果

执行结果如图 10.15 所示。

图 10.35 异步读取文件命令提示符界面

2. 同步

同步读取没有回调函数，输入欲读取的文件名，所读内容作为返回值传给调用者。

例 10-3-1-2 一个简单同步的读取文件示例。

（1）创建文件 10-3-1-2.js

源程序如下。

```
var fs=require("fs");
//同步读取
console.log("启动读取文件");
var data=fs.readFileSync('例 10-3-1-1.txt');
console.log("------------------------------------");
console.log("同步读取: "+data.toString());
console.log("------------------------------------");
console.log("程序执行完毕。");
```

（2）执行结果

执行结果如图 10.36 所示。



图 10.36 同步读取文件命令提示符界面

接下来具体了解 Node.js 文件系统的方法。

10.3.2 文件操纵

1. 打开文件

（1）异步模式下打开文件的语法

```
fs.open(path, flags[, mode], callback)
```

（2）参数使用说明

path：文件的路径。

flags：文件打开的行为。具体值详见表 10.1。

mode：设置文件模式（权限），文件创建默认权限为 0666（可读，可写）。

callback：回调函数，带有两个参数，如 callback(err, fd)。

表 10.1 flags 参数值

Flag	描　述
r	以读取模式打开文件。如果文件不存在,则抛出异常
r+	以读写模式打开文件。如果文件不存在,则抛出异常
rs	以同步的方式读取文件
rs+	以同步的方式读取和写入文件
w	以写入模式打开文件,如果文件不存在,则创建
wx	类似'w',如果文件路径存在,则文件写入失败
w+	以读写模式打开文件,如果文件不存在,则创建
wx+	类似'w+',如果文件路径存在,则文件读写失败
a	以追加模式打开文件,如果文件不存在,则创建
ax	类似'a',如果文件路径存在,则文件追加失败
a+	以读取追加模式打开文件,如果文件不存在,则创建
ax+	类似'a+',如果文件路径存在,则文件读取追加失败

(3) 示例

源程序如下所示。

```
var fs=require("fs");
//异步打开文件
console.log("准备打开文件!");
fs.open('input.txt', 'r+', function(err, fd) {
    if (err) return console.error(err);
    console.log("文件打开成功!");
});
```

2. 获取文件信息

(1) 异步模式获取文件信息的语法

```
fs.stat(path, callback)
```

(2) 参数使用说明

path:文件路径。

callback:回调函数,带有两个参数,如:(err,stats),stats 是 fs.stats 对象。

执行 fs.stat(path)后,会将 stats 类的实例返回给其回调函数,可以通过 stats 类中的提供方法判断文件的相关属性。例如,判断是否为文件的语句如下。

```
var fs=require('fs');
fs.stat('/Users/liuht/code/itbilu/demo/fs.js', function (err, stats) {
    console.log(stats.isFile());        //true
})
```

除判断文件之外，回传的 stats 对象还有其他方法，如表 10.2 所示。

表 10.2 stats 类中的方法

方 法	描 述
stats.isFile()	如果是文件，则返回 true，否则返回 false
stats.isDirectory()	如果是目录，则返回 true，否则返回 false
stats.isBlockDevice()	如果是块设备，则返回 true，否则返回 false
stats.isCharacterDevice()	如果是字符设备，则返回 true，否则返回 false
stats.isSymbolicLink()	如果是软链接，则返回 true，否则返回 false
stats.isFIFO()	如果是 FIFO，则返回 true，否则返回 false。FIFO 是 UNIX 中一种特殊类型的命令管道
stats.isSocket()	如果是 Socket，则返回 true，否则返回 false

(3) 示例

源程序如下所示。

```
var fs=require("fs");
console.log("准备打开文件!");
fs.stat('input.txt', function (err, stats) {
    if (err) return console.error(err);
    console.log(stats);
    console.log("读取文件信息成功!");
    //检测文件类型
    console.log("是否为文件(isFile)? "+stats.isFile());
    console.log("是否为目录(isDirectory)? "+stats.isDirectory());
});
```

3. 写入/创建文件

(1) 异步模式下写入文件的语法

```
fs.writeFile(file, data[, options], callback)
```

如果文件存在，该方法写入的内容会覆盖旧的文件内容。否则创建文件。

(2) 参数使用说明

file：文件名或文件描述符。

data：要写入文件的数据，可以是 String(字符串)或 Buffer(流)对象。

options：该参数是一个对象，包含{encoding, mode, flag}。默认编码为 UTF8，模式为 0666，flag 为'w'。

callback：回调函数，回调函数只包含错误信息参数(err)，写入失败时返回。

(3) 示例

源程序如下所示。

```
var fs=require("fs");
console.log("准备写入文件");
fs.writeFile('input.txt', '我是通过写入的文件内容！ ',  function(err) {
    if (err) return console.error(err);
    console.log("数据写入成功!");
    console.log("--------我是分割线-------------")
    console.log("读取写入的数据!");
    fs.readFile('input.txt', function (err, data) {
        if (err) return console.error(err);
        console.log("异步读取文件数据: "+data.toString());
    });
});
```

4. 读取文件

（1）异步模式下读取文件的语法格式

```
fs.read(fd, buffer, offset, length, position, callback)
```

该方法使用了文件描述符来读取文件。

（2）参数使用说明

fd：通过 fs.open() 方法返回的文件描述符。

buffer：数据写入的缓冲区。

offset：缓冲区写入的写入偏移量。

length：要从文件中读取的字节数。

position：文件读取的起始位置，如果 position 的值为 null，则会从当前文件指针的位置读取。

callback：回调函数，有 3 个参数 err、bytesRead 和 buffer，err 为错误信息，bytesRead 表示读取的字节数，buffer 为缓冲区对象。

5. 关闭文件

（1）异步模式下关闭文件的语法

```
fs.close(fd, callback)
```

该方法使用了文件描述符来读取文件。

（2）参数使用说明

fd：通过 fs.open() 方法返回的文件描述符。

callback：回调函数，没有参数。

6. 截取文件

（1）异步模式下截取文件的语法

```
fs.ftruncate(fd, len, callback)
```

该方法使用了文件描述符来读取文件。

(2) 参数使用说明

fd：通过 fs.open() 方法返回的文件描述符。

len：文件内容截取的长度。

callback：回调函数，没有参数。

7. 删除文件

(1) 语法格式

```
fs.unlink(path, callback)
```

(2) 参数使用说明

path：文件路径。

callback：回调函数，没有参数。

10.3.3 目录操纵

1. 创建目录

(1) 语法格式

```
fs.mkdir(path[, mode], callback)
```

(2) 参数使用说明

path：文件路径。

mode：设置目录权限，默认为 0777。

callback：回调函数，没有参数。

(3) 示例

```
var fs=require("fs");
console.log("创建目录 /tmp/test/");
fs.mkdir("/tmp/test/",function(err){
    if (err) {
        return console.error(err);
    }
    console.log("目录创建成功。");
});
```

2. 读取目录

(1) 语法格式

```
fs.readdir(path, callback)
```

(2) 参数使用说明

path：文件路径。

callback：回调函数，回调函数带有两个参数 err、files，err 为错误信息，files 为目录下的文件数组列表。

(3) 示例

```
var fs=require("fs");
console.log("查看 /tmp 目录");
fs.readdir("/tmp/",function(err, files){
    if (err) return console.error(err);
    files.forEach( function (file){
        console.log( file );
    });
});
```

3. 删除目录

(1) 删除目录的语法

```
fs.rmdir(path, callback)
```

(2) 参数使用说明

path：文件路径。

callback：回调函数，没有参数。

(3) 示例

```
var fs=require("fs");
//执行前创建一个空的 /tmp/test 目录
console.log("准备删除目录 /tmp/test");
fs.rmdir("/tmp/test",function(err){
    if (err) return console.error(err);
    console.log("读取 /tmp 目录");
    fs.readdir("/tmp/",function(err, files){
        if (err) return console.error(err);
        files.forEach( function (file){
            console.log( file );
        });
    });
});
```

10.4 事　　件

10.4.1 回调函数

Node.js 异步编程的直接体现就是回调。异步编程依托于回调实现，但不能说使用

了回调后程序就异步化了。回调函数在完成任务后就会被调用，Node 使用了大量的回调函数，Node 所有 API 都支持回调函数。

在例 10-3-1-1 中，一边读取文件，一边执行其他命令。文件读取完成后，将文件内容作为回调函数的参数返回。这样，执行代码时就没有阻塞或等待文件 I/O 操作。这就极大地提高了 Node.js 的性能，可以处理大量的并发请求。

例 10-3-1-1 和例 10-3-1-2 两个实例讲述了阻塞与非阻塞调用的不同。第二个实例在文件读取完后才执行完程序。第一个实例不需要等待文件读取完，在读取文件时就可以同时执行接下来的代码，极大地提高了程序的性能。

因此，阻塞是按顺序执行的，而非阻塞是不需要按顺序的。所以，如果需要处理回调函数的参数，就需要写在回调函数内。

10.4.2 事件循环

Node.js 是单进程单线程应用程序，但是通过事件和回调支持并发，所以性能非常高。Node.js 的每一个 API 都是异步的，并作为一个独立线程运行，使用异步函数调用，并处理并发。Node.js 所有的事件机制基本上都是用设计模式中观察者模式实现。

Node.js 单线程类似进入一个 while(true)的事件循环，直到没有事件观察者退出。每个异步事件都生成一个事件观察者，如果有事件发生，就调用该回调函数。

Node.js 使用事件驱动模型，如图 10.37 所示。当 Web Server 接收到请求，就关闭它，然后进行处理，最后去服务下一个 Web 请求。这个请求完成后，被放回处理队列，当到达队列开头，这个结果被返回给用户。

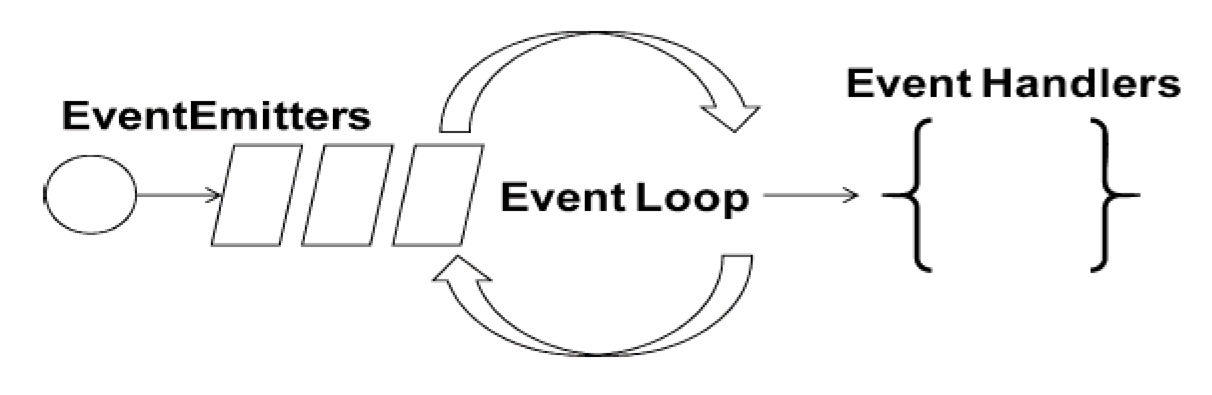


图 10.37　事件驱动模型

这个模型非常高效，可扩展性非常强，因为 Web Server 一直接受请求而不等待任何读写操作(这也被称为非阻塞式 I/O 或者事件驱动 I/O)。

事件驱动模型中会生成一个主循环来监听事件，检测到事件时触发回调函数。

整个事件驱动的流程就是这么实现的，非常简洁。有点类似于观察者模式，事件相当于一个主题，而所有注册到这个事件上的处理函数相当于观察者。

Node.js 有多个内置的事件，可以通过引入 events 模块，并通过实例化 EventEmitter 类来绑定和监听事件，如例 10-4-2-1 所示。

例 10-4-2-1　事件绑定处理示例。

(1) 创建 10-4-2-1.js 文件

文件内容如下。

```
var events=require('events'); //引入 events 模块
var eventEmitter=new events.EventEmitter();     //创建 eventEmitter 对象
//创建事件处理程序
var connectHandler=function connected() {
    console.log('连接成功。');
    eventEmitter.emit('data_received');          //触发 data_received 事件
}
eventEmitter.on('connection', connectHandler); //绑定 connection 事件处理程序
//使用匿名函数绑定 data_received 事件
eventEmitter.on('data_received', function(){
    console.log('数据接收成功。');
});
eventEmitter.emit('connection');                //触发 connection 事件
console.log("程序执行完毕。");
```

(2) 执行结果

执行结果如图 10.38 所示。

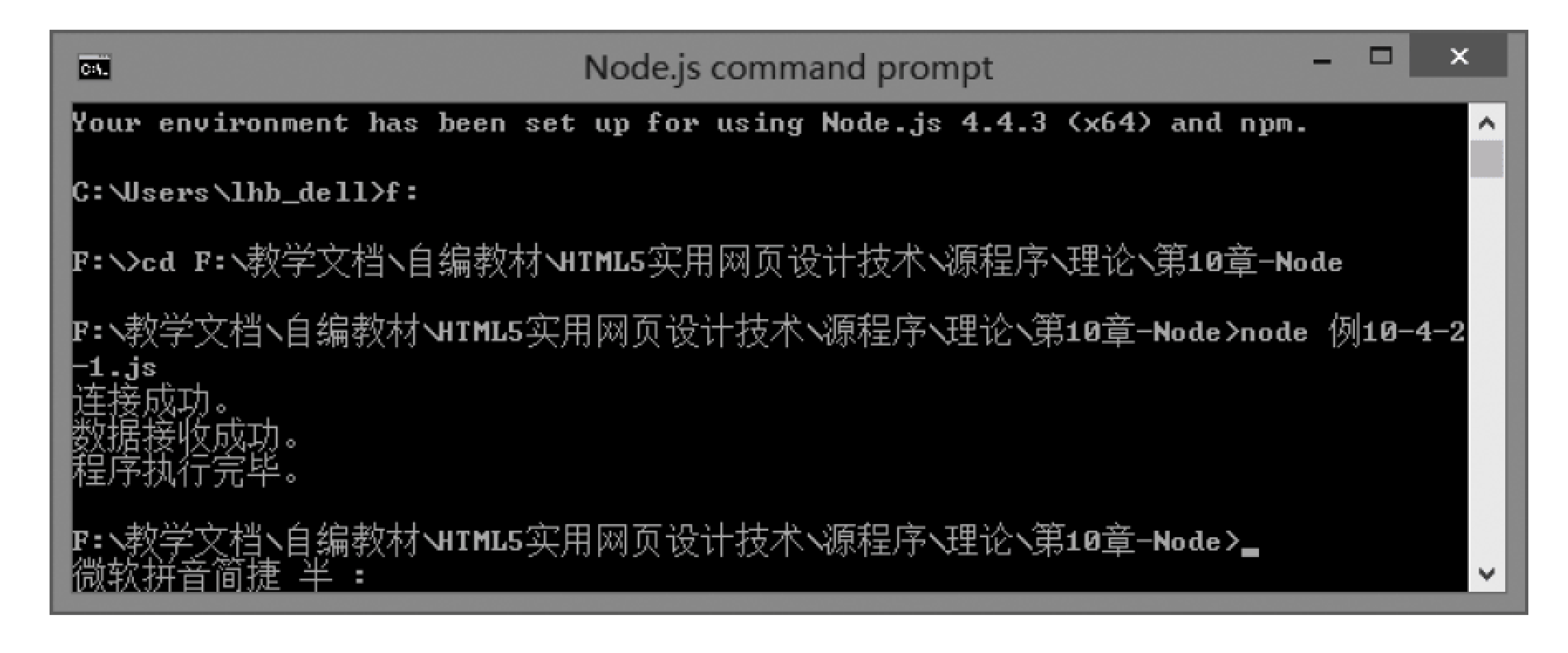


图 10.38 例 10-4-2-1.js 的运行结果

10.4.3 EventEmitter

Node.js 所有的异步 I/O 操作完成时都会发送一个事件到事件队列。Node.js 里面的许多对象都会分发事件：一个 net.Server 对象会在每次有新连接时分发一个事件，一个 fs.readStream 对象会在文件被打开时发出一个事件。所有这些产生事件的对象都是 events.EventEmitter 的实例。

events 模块只提供了一个对象：events.EventEmitter。EventEmitter 的核心就是事件触发与事件监听器功能的封装。

通过 require("events");来访问 events 模块，进而实例化 EventEmitter，如下所示。

```
//引入 events 模块
var events=require('events');
//创建 eventEmitter 对象
var eventEmitter=new events.EventEmitter();
```

如果 EventEmitter 对象在实例化时发生错误，会触发 error 事件。当添加新的监听器时，newListener 事件会触发，当监听器被移除时，removeListener 事件被触发。

例 10-4-3-1 一个简单的 EventEmitter 的运用。

（1）例 10-4-3-1.js 的内容

```
var EventEmitter=require('events').EventEmitter;
var event=new EventEmitter();
event.on('some_event', function () { console.log('some_event 已处理');});
setTimeout(function () { event.emit('some_event');}, 1000);
```

（2）例 10-4-3-1.js 的执行结果。

执行结果如图 10.39 所示。

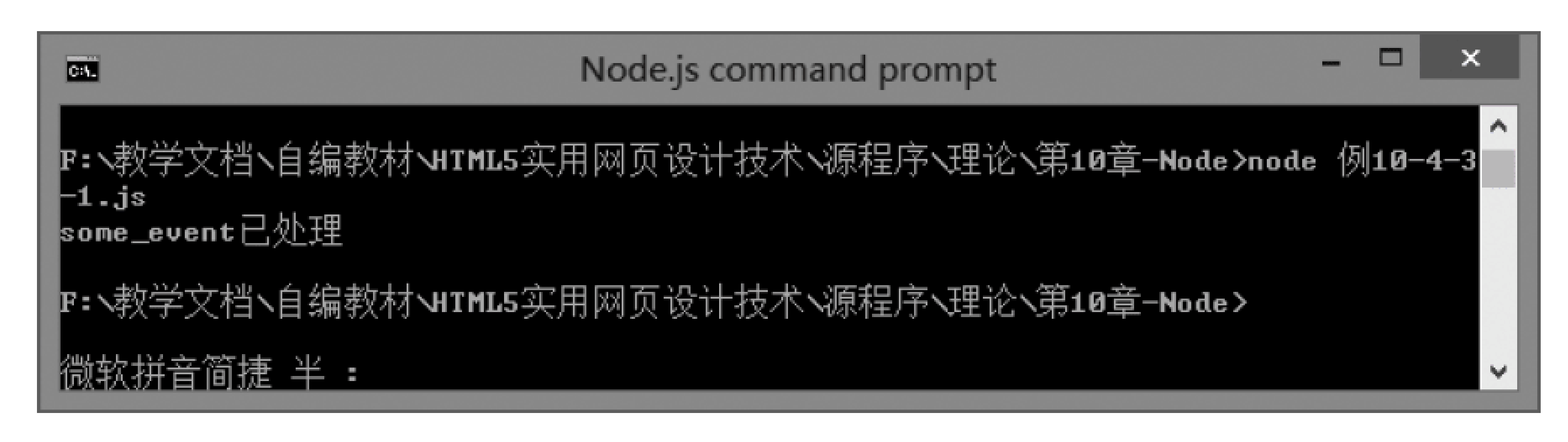


图 10.39 例 10-4-3-1.js 的运行结果

（3）例 10-4-3-1.js 的程序分析

运行这段代码，1 秒后控制台输出了"some_event 已处理"。其原理是 event 对象注册了事件 some_event 的一个监听器，然后通过 setTimeout，在 1000ms 以后向 event 对象发送事件 some_event，此时会调用 some_event 的监听器。

1. 事件参数

EventEmitter 的每个事件由一个事件名和若干个参数组成，事件名是一个字符串，通常表达一定的语义。对于每个事件，EventEmitter 支持若干个事件监听器。当事件触发时，注册到这个事件的事件监听器被依次调用，事件参数作为回调函数参数传递。

例 10-4-3-1-1 事件参数示例。

（1）建立例 10-4-3-1-1.js 文件

文件内容如下。

```
var events=require('events');
var emitter=new events.EventEmitter();
emitter.on('someEvent', function(arg1, arg2) { console.log('listener1', arg1, arg2); });
emitter.on('someEvent', function(arg1, arg2) { console.log('listener2', arg1, arg2); });
emitter.emit('someEvent', 'arg1 参数', 'arg2 参数');
```

（2）执行例 10-4-3-1-1.js

执行结果如图 10.40 所示。

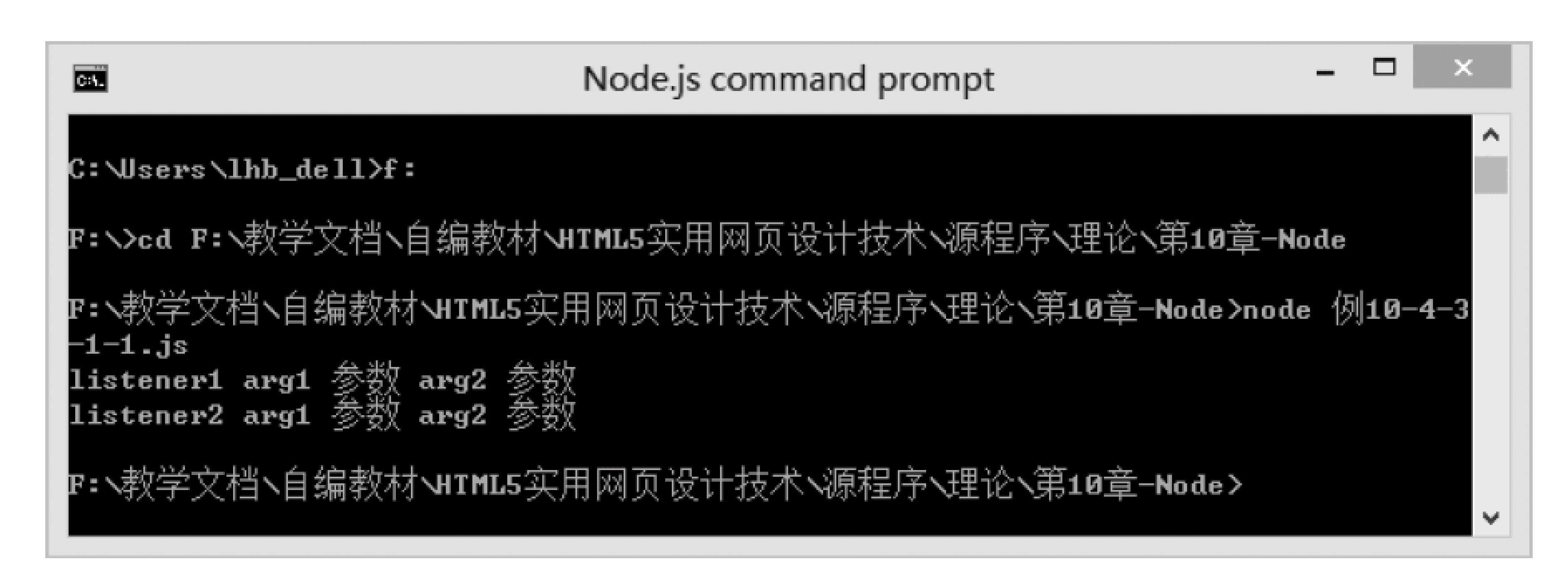


图 10.40　例 10-4-3-1-1.js 的运行结果

(3) 例 10-4-3-1-1.js 的程序分析

以上例子中，emitter 为事件 someEvent 注册了两个事件监听器，然后触发了 someEvent 事件。从运行结果中可以看到两个事件监听器回调函数被先后调用。这就是 EventEmitter 最简单的用法。

2. EventEmitter 对象方法

EventEmitter 提供了多个方法，如 on 和 emit。on 函数用于绑定事件函数，emit 属性用于触发一个事件。常用的方法如表 10.3 所示。

表 10.3　EventEmitter 对象方法

序号	方　法	描　述
1	addListener(event,listener)	为指定事件添加一个监听器到监听器数组的尾部
2	on(event,response)	创建监听器，并将 event 事件和响应函数 response 相关联。event 为一个字符串，表示事件名字。response 为一个响应函数
3	once(event,response)	创建单次监听器，并将 event 事件和响应函数 response 相关联。event 为一个字符串，表示事件名字。response 为一个响应函数。监听器最多只会触发一次，触发后立刻解除该监听器
4	removeListener(event,response)	移除指定事件的某个监听器，监听器必须是该事件已经注册过的监听器。它接受两个参数，第一个是事件名称，第二个是回调函数名称
5	removeAllListeners([event])	移除所有事件的所有监听器，如果指定事件，则移除指定事件的所有监听器
6	setMaxListeners(n)	默认情况下，如果添加的监听器超过 10 个 EventEmitters 就会输出警告信息。setMaxListeners 函数用于提高监听器的默认限制的数量
7	listeners(event)	返回指定事件的监听器数组
8	emit(event,[arg1],[arg2],[...])	按参数的顺序执行每个监听器，如果事件有注册监听，返回 true，否则返回 false

3. 实例演示

(1) 创建例 10-4-3-1-3.js 文件

文件内容如下。

```
var events=require('events');
var eventEmitter=new events.EventEmitter();
//监听器 #1
var listener1=function listener1() {  console.log('监听器 listener1 执行。'); }
//监听器 #2
var listener2=function listener2() {  console.log('监听器 listener2 执行。'); }
//绑定 connection 事件,处理函数为 listener1
eventEmitter.addListener('connection', listener1);
//绑定 connection 事件,处理函数为 listener2
eventEmitter.on('connection', listener2);
var eventListeners = require ('events').EventEmitter.listenerCount (eventEmitter,'
connection');
console.log(eventListeners+"个监听器监听连接事件。");
//处理 connection 事件
eventEmitter.emit('connection');
//移除监绑定的 listener1 函数
eventEmitter.removeListener('connection', listener1);
console.log("listener1 不再受监听。");
//触发连接事件
eventEmitter.emit('connection');
eventListeners=require('events').EventEmitter.listenerCount(eventEmitter,'connection
');
console.log(eventListeners+"个监听器监听连接事件。");
console.log("程序执行完毕。");
```

(2) 执行结果

执行结果如图 10.41 所示。

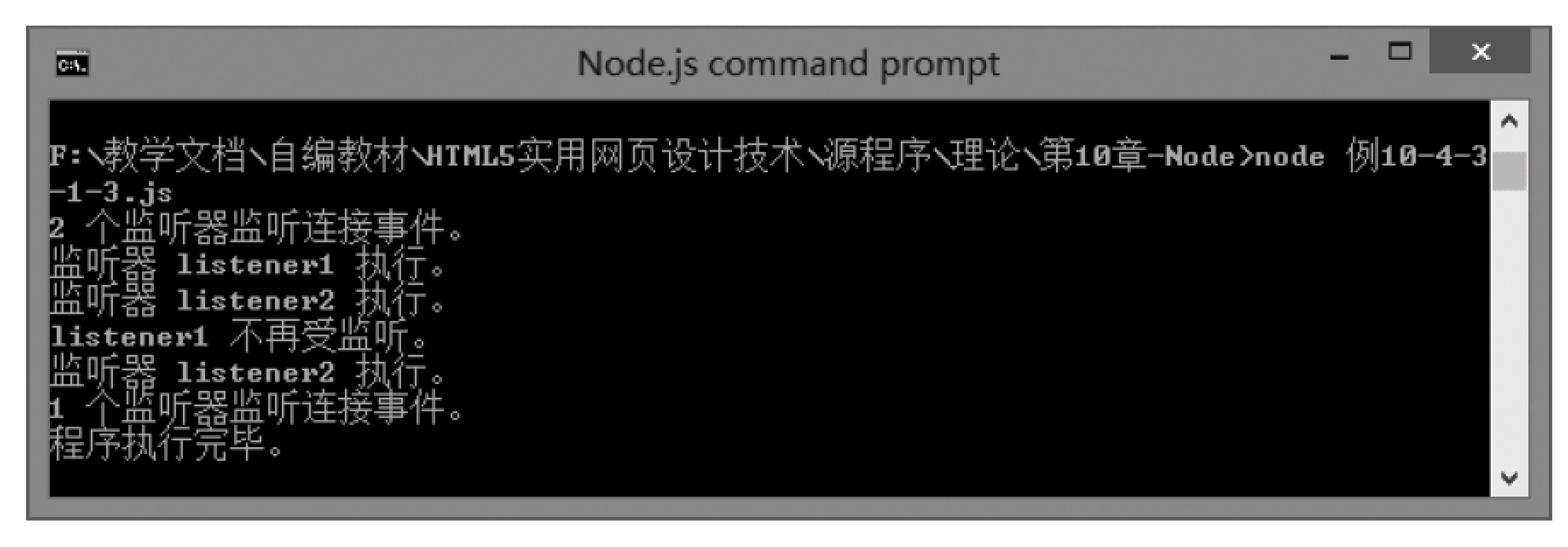


图 10.41　例 10-4-3-1-3.js 的运行结果

10.5 实验任务

1. 实验题目

基于 Node.js 和 jQueryMobile 开发手机版画图程序。

2. 程序功能

浏览器端页面功能与 9.3 的程序功能相同。利用 Node.js 的文件记录用户选择的背景色、线宽和运动速度。下次登录时自动启用用户的最近一次设置参数。

3. 实验目的

(1) 掌握 Node.js 的 Web 服务器发布技术。
(2) 掌握 Node.js 的文件管理技术。
(3) 掌握 Node.js 的事件定义及处理技术。

4. 实验类型

综合设计。

5. 实验要求

(1) 新登录用户，创建参数文件。每个用户一个参数文件。
(2) 服务器端的命令提示符界面报告用户登录和更改设置情况。

6. 实验环境

(1) 服务器端
① 计算机：PC、内存 8GB、主频 1.8GHz 及以上、硬盘 500GB 及以上。
② 操作系统：Windows XP、Windows 7、Windows 8、Windows 10。
③ 开发环境：Visual Studio 2010。
④ Web 服务器：IIS。
⑤ 浏览器：Chrome 或 QQ 浏览器。
(2) 浏览器端
① 手机、PAD。
② Chrome 或 Safari。
(3) 网络环境
无线上网。

7. 实验原理

绘出浏览器与 Node.js 服务器的同步关系图。

8. 遇到的问题及解决办法

9. 运行结果

给出所完成任务功能的屏幕截图。

参 考 文 献

[1] 赵峰. 网页设计与制作：HTML5+CSS+JavaScript(第2版)[M]. 北京：清华大学出版社，2013.

[2] 杨东昱. HTML5+CSS3精致范例辞典[M]. 北京：清华大学出版社，2013.

[3] 陈惠贞. HTML5、CSS3、RWD、jQuery Mobile跨设备网页设计[M]. 北京：清华大学出版社，2016.

[4] 王寅峰. HTML5跨平台开发基础与实战[M]. 北京：高等教育出版社，2014.

[5] 陈承欢. HTML5+CSS3网页美化与布局任务驱动式教程(第2版)[M]. 北京：高等教育出版社，2015.

[6] 马科. HTML5 App商业开发实战教程：基于WeX5可视化开发平台[M]. 北京：高等教育出版社，2016.

[7] LaGrone B. 响应式Web设计：HTML5和CSS3实践指南[M]. 北京：机械工业出版社，2014.

[8] Wang V，Salim F，Moskovits P. HTML5 WebSocket权威指南[M]. 姚军，等译. 北京：机械工业出版社，2014.

[9] Lengstorf J，Leggetter P. 构建实时Web应用：基于HTML5 WebSocket、PHP和jQuery[M]. 肖智清，译. 北京：机械工业出版社，2013.

[10] 刘德山，章增安，孙美乔. HTML5+CSS3 Web前端开发技术[M]. 北京：人民邮电出版社，2016.

[11] 刘欢. HTML5基础知识、核心技术与前沿案例[M]. 北京：人民邮电出版社，2016.

[12] Cook D. HTML5数据推送应用开发[M]. 刘帅，译. 北京：人民邮电出版社，2014.

[13] Woods S. HTML5触摸界面设计与开发[M]. 谷岳，等译. 北京：人民邮电出版社，2014.

[14] 马骏. HTML5与ASP.NET程序设计教程(第2版)[M]. 北京：人民邮电出版社，2012.